*You are currently looking at **version 1.5** 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 (https://www.coursera.org/learn/python-machine-learning/resources/bANLa) course resource.*

Assignment 2

In this assignment you'll explore the relationship between model complexity and generalization performance, by adjusting key parameters of various supervised learning models. Part 1 of this assignment will look at regression and Part 2 will look at classification.

Part 1 - Regression

First, run the following block to set up the variables needed for later sections.

In [18]:

```
import numpy as np
import pandas as pd
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
print(x.shape)
print(y.shape)

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

# You can use this function to help you visualize the dataset by
# plotting a scatterplot of the data points
# in the training and test sets.
def part1_scatter():
    import matplotlib.pyplot as plt
    %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);


# NOTE: Uncomment the function below to visualize the data, but be sure
# to **re-comment it before submitting this assignment to the autograder**.
#part1_scatter()
```

```
(15,)
(15,)
```

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 [50]:

```
def answer_one():
    from sklearn.linear_model import LinearRegression
    from sklearn.preprocessing import PolynomialFeatures

    # Your code here
    # I will proceed without normalising the data
    result = np.zeros(400).reshape(4,100)
    for idx, deg in enumerate([1,3,6,9]):
        poly = PolynomialFeatures(degree = deg)
        #create the respective training and test sets
        # Need to transpose X_train and query_pts for the poly fit to work
        X_train_poly = poly.fit_transform(X_train[None].T)
        query_pts = poly.transform(np.linspace(0,10,100)[None].T) # do not fit t
ransform to the query_pts!
        #print((X_train_poly.shape, query_pts.shape))
        # Train the model
        linreg = LinearRegression().fit(X_train_poly, y_train)
        # Make 100 predictions on this model for this degree

        # Insert this into the result list
        result[idx,:] = linreg.predict(query_pts)

    return result# Return your answer
#answer_one()
```

Out[50]:

```
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,
```

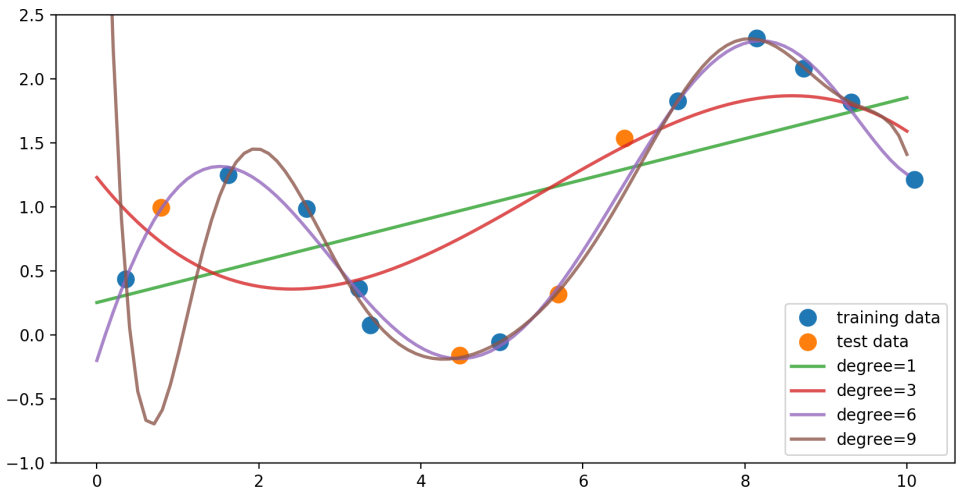
```
    1.76093671e+00,   1.78032933e+00,   1.79800506e+00,
    1.81388438e+00,   1.82788778e+00,   1.83993575e+00,
    1.84994877e+00,   1.85784732e+00,   1.86355189e+00,
    1.86698296e+00,   1.86806103e+00,   1.86670656e+00,
    1.86284006e+00,   1.85638200e+00,   1.84725286e+00,
    1.83537314e+00,   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,  -3.95192724e-03,   1.79851752e-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,  -3.41690366e-02,
   -7.71657636e-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,  -5.50696232e-02,
   -6.01973198e-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],
[  6.79502285e+00,   4.14319957e+00,   2.23123322e+00,
    9.10495532e-01,   5.49803315e-02,  -4.41344457e-01,
   -6.66950444e-01,  -6.94942887e-01,  -5.85049614e-01,
   -3.85418417e-01,  -1.34236065e-01,   1.38818559e-01,
    4.11275202e-01,   6.66715442e-01,   8.93747460e-01,
    1.08510202e+00,   1.23683979e+00,   1.34766069e+00,
    1.41830632e+00,   1.45104724e+00,   1.44924694e+00,
    1.41699534e+00,   1.35880444e+00,   1.27935985e+00,
    1.18332182e+00,   1.07516995e+00,   9.59086410e-01,
    8.38872457e-01,   7.17893658e-01,   5.99049596e-01,
    4.84764051e-01,   3.76992063e-01,   2.77240599e-01,
    1.86599822e-01,   1.05782272e-01,   3.51675757e-02,
   -2.51494865e-02,  -7.53094019e-02,  -1.15638484e-01,
   -1.46600958e-01,  -1.68753745e-01,  -1.82704910e-01,
   -1.89076542e-01,  -1.88472636e-01,  -1.81452388e-01,
   -1.68509141e-01,  -1.50055083e-01,  -1.26411638e-01,
   -9.78053923e-02,  -6.43692604e-02,  -2.61485139e-02,
    1.68888091e-02,   6.48376626e-02,   1.17838541e-01,
```

```
        1.76057485e-01,   2.39664260e-01,   3.08809443e-01,
        3.83601186e-01,   4.64082407e-01,   5.50209170e-01,
        6.41830991e-01,   7.38673768e-01,   8.40326006e-01,
        9.46228923e-01,   1.05567100e+00,   1.16778742e+00,
        1.28156471e+00,   1.39585100e+00,   1.50937183e+00,
        1.62075165e+00,   1.72854097e+00,   1.83124862e+00,
        1.92737898e+00,   2.01547331e+00,   2.09415458e+00,
        2.16217465e+00,   2.21846257e+00,   2.26217273e+00,
        2.29273094e+00,   2.30987668e+00,   2.31369926e+00,
        2.30466539e+00,   2.28363551e+00,   2.25186569e+00,
        2.21099186e+00,   2.16299265e+00,   2.11012671e+00,
        2.05484041e+00,   1.99964089e+00,   1.94692956e+00,
        1.89879060e+00,   1.85672836e+00,   1.82134774e+00,
        1.79197049e+00,   1.76618058e+00,   1.73929091e+00,
        1.70372341e+00,   1.64829405e+00,   1.55739372e+00,
        1.41005558e+00]])
```

In [39]:

```
# 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):
    import matplotlib.pyplot as plt
    %matplotlib notebook
    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 one tuple of numpy arrays `(r2_train, r2_test)`. Both arrays should have shape `(10,)`

In [53]:

```
def answer_two():
    from sklearn.linear_model import LinearRegression
    from sklearn.preprocessing import PolynomialFeatures
    from sklearn.metrics.regression import r2_score

    # Your code here
    r2_train, r2_test = np.zeros(10), np.zeros(10)
    #print(r2_train.shape, r2_test.shape)
    for deg in range(10):
        poly = PolynomialFeatures(degree = deg)
        #print(X_train.shape)
        X_train_poly = poly.fit_transform(X_train[None].T)  # This creates a mat
rix for the variable X_train
        #print(X_train_poly.shape)
        X_test_poly = poly.transform(X_test[None].T) # Do not fit transform to t
he test set
        #print(X_test_poly.shape)
        linreg = LinearRegression().fit(X_train_poly, y_train)
        r2_train[deg] = linreg.score(X_train_poly, y_train)
        r2_test[deg] = linreg.score(X_test_poly, y_test)

    return (r2_train, r2_test) # Your answer here
#answer_two()
```

Out[53]:

```
(array([ 0.        ,  0.42924578,  0.4510998 ,  0.58719954,  0.91941
945,
         0.97578641,  0.99018233,  0.99352509,  0.99637545,  0.99803
706]),
 array([-0.47808642, -0.45237104, -0.06856984,  0.00533105,  0.73004
943,
         0.87708301,  0.9214094 ,  0.92021504,  0.63247951, -0.64525
377]))
```

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 between degree level and R^2. Remember to comment out the import matplotlib line before submission.

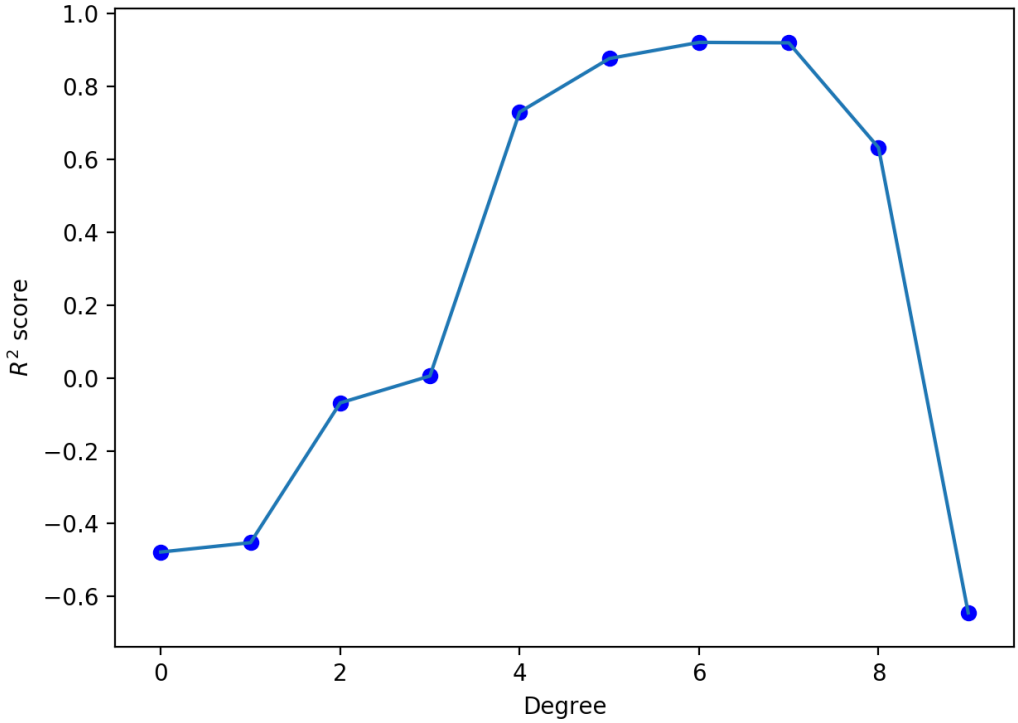
This function should return one tuple with the degree values in this order: `(Underfitting, Overfitting, Good_Generalization)`. *There might be multiple correct solutions, however, you only need to return one possible solution, for example, (1,2,3).*

In [79]:

```
# Function to plot R^2 values
def plot_R2_values_wrt_degree():
    r2_test = answer_two()[1]
    %matplotlib notebook
    import matplotlib.pyplot as plt
    plt.figure(figsize= (7,5))
    plt.plot(range(10), r2_test)
    plt.scatter(range(10), r2_test, c = "b") #overlay another scatter plot
    plt.xlabel('Degree')
    plt.ylabel('$R^2$ score')
    plt.show()

def answer_three():
    #plot_R2_values_wrt_degree()
    # Your code here
    # Okay, so I'm just going to use my plot above to make a deduction. When deg
ree = 0, the model is most likely
    # to underfit the data since polynomial regression at degree 0 is essentiall
y just the bias term, and hence too
    # simple to model the underlying distribution. The overfitting generally occ
urs after the maximum point in the graph
    # shown above, which is why models with degree 8 and 9 have poorer R^2 value
s on the test set - as the model has
    # began to overfit the training data.
    # At high R^2 values on the test set, this is a good generalisation of the u
nderlying distribution.

    return (0,9,6)# Return your answer
#answer_three()
```




```
Out[79]:
```

```
(0, 9, 6)
```

Question 4

Training models on high degree polynomial features can result in overly complex models that overfit, so we often use regularized versions of the model to constrain model complexity, as we saw with Ridge and Lasso linear regression.

For this question, train two models: a non-regularized LinearRegression model (default parameters) and a regularized Lasso Regression model (with parameters `alpha=0.01`, `max_iter=10000`) both on polynomial features of degree 12. Return the R^2 score for both the LinearRegression and Lasso model's test sets.

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

In [57]:

```
def answer_four():
    from sklearn.preprocessing import PolynomialFeatures
    from sklearn.linear_model import Lasso, LinearRegression
    from sklearn.metrics.regression import r2_score

    # Your code here
    poly = PolynomialFeatures(degree = 12)
    X_train_poly = poly.fit_transform(X_train[None].T)
    X_test_poly = poly.transform(X_test[None].T)
    # Model with no regularisation
    noreg = LinearRegression().fit(X_train_poly, y_train)
    # model with regularisation
    reg = Lasso(alpha = 0.01, max_iter=10000).fit(X_train_poly, y_train)


    return (noreg.score(X_test_poly, y_test), reg.score(X_test_poly, y_test)) #
 Your answer here
answer_four()
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/linear_model/coordina
te_descent.py:484: ConvergenceWarning: Objective did not converge. Y
ou might want to increase the number of iterations. Fitting data wit
h very small alpha may cause precision problems.
  ConvergenceWarning)
```

Out[57]:

```
(-4.3120017974975458, 0.8406625614750235)
```

Part 2 - Classification

Here's an application of machine learning that could save your life! For this section of the assignment we will be working with the UCI Mushroom Data Set (http://archive.ics.uci.edu/ml/datasets/Mushroom?ref=datanews.io) stored in `readonly/mushrooms.csv`. The data will be used to train 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 [58]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split


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

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

# use the variables X_train2, y_train2 for Question 5
X_train2, X_test2, y_train2, y_test2 = train_test_split(X_mush, y_mush, random_s
tate=0)

# For performance reasons in Questions 6 and 7, we will create a smaller version
of the
# entire mushroom dataset for use in those questions.  For simplicity we'll just
re-use
# the 25% test split created above as the representative subset.
#
# Use the variables X_subset, y_subset for Questions 6 and 7.
X_subset = X_test2
y_subset = y_test2
```

Question 5

Using `X_train2` and `y_train2` 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?

As a reminder, the feature names are available in the `X_train2.columns` property, and the order of the features in `X_train2.columns` matches the order of the feature importance values in the classifier's `feature_importances_` property.

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

Note: remember that you also need to set random_state in the DecisionTreeClassifier.

In [78]:

```
def answer_five():
    from sklearn.tree import DecisionTreeClassifier
    # Your code here
    dtc = DecisionTreeClassifier().fit(X_train2, y_train2)
    # Get the indices of the top 5 elements.
    # Take the negative of the array - the largest values in the feature importa
nces array
    # will become the most negative values (hence the smallest) in the negative
 array.
    indices = (-(dtc.feature_importances_)).argsort()[:5]
    # We have obtained the column indices in the variable 'indices' of the top 5
most important features

    # Now return the list with the column NAMES instead.
    return [X_train2.columns[x] for x in indices]
#answer_five()
```

Out[78]:

```
['odor_n', 'stalk-root_c', 'stalk-root_r', 'spore-print-color_r', 'o
dor_l']
```

Question 6

For this question, we're going to 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. Recall that the validation_curve function, in addition to taking an initialized unfitted classifier object, takes a dataset as input and does its own internal train-test splits to compute results.

Because creating a validation curve requires fitting multiple models, for performance reasons this question will use just a subset of the original mushroom dataset: please use the variables X_subset and y_subset as input to the validation curve function (instead of X_mush and y_mush) to reduce computation time.

The initialized unfitted classifier object we'll be using is a Support Vector Classifier with radial basis kernel. So your first step is to create an `SVC` object 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.

With this classifier, and the dataset in X_subset, y_subset, 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)`). Recall that you can specify what scoring metric you want validation_curve to use by setting the "scoring" parameter. In this case, we want to use "accuracy" as the scoring metric.

For each level of `gamma`, `validation_curve` will fit 3 models on different subsets of the data, returning two 6x3 (6 levels of gamma x 3 fits per level) arrays of the scores for the training and test sets.

Find the mean score across the three 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 one tuple of numpy arrays `(training_scores, test_scores)` where each array in the tuple has shape `(6,)`.

In [71]:

```
def answer_six():
    from sklearn.svm import SVC
    from sklearn.model_selection import validation_curve

    # Your code here
    # Create training and test data based on this subset
    X_train3, y_train3, X_test3, y_test3 = train_test_split(X_subset, y_subset,
                                                            random_state = 0)
    #Range of values for gamma
    gamma_range = np.logspace(-4,1,6)
    train_scores, test_scores = validation_curve(SVC(kernel = 'rbf', C = 1), # d
efault values but I'm just gonna
                                                 X_subset, y_subset,         #wr
ite it for understanding purposes.
                                                param_name = 'gamma',
                                                param_range = gamma_range, cv =
3)
    #print(train_scores)
    #print(test_scores)
    train_scores = np.array(train_scores); test_scores = np.array(test_scores)
    avg_train = np.array([np.mean(x) for x in train_scores])
    avg_test = np.array([np.mean(x) for x in test_scores])
    return (avg_train, avg_test) # Your answer here
#answer_six()
```

Out[71]:

```
(array([ 0.56647847,  0.93155951,  0.99039881,  1.        ,  1.
,  1.        ]),
 array([ 0.56768547,  0.92959558,  0.98965952,  1.        ,  0.99507
994,
         0.52240279]))
```

Question 7

Based on the scores from question 6, what gamma value corresponds to a model that is underfitting (and has the worst test set accuracy)? What gamma value corresponds to a model that is overfitting (and has the worst test set accuracy)? What choice of gamma would be the best choice for a model with good generalization performance on this dataset (high accuracy on both training and test set)?

Hint: Try plotting the scores from question 6 to visualize the relationship between gamma and accuracy. Remember to comment out the import matplotlib line before submission.

This function should return one tuple with the degree values in this order: `(Underfitting, Overfitting, Good_Generalization)` Please note there is only one correct solution.

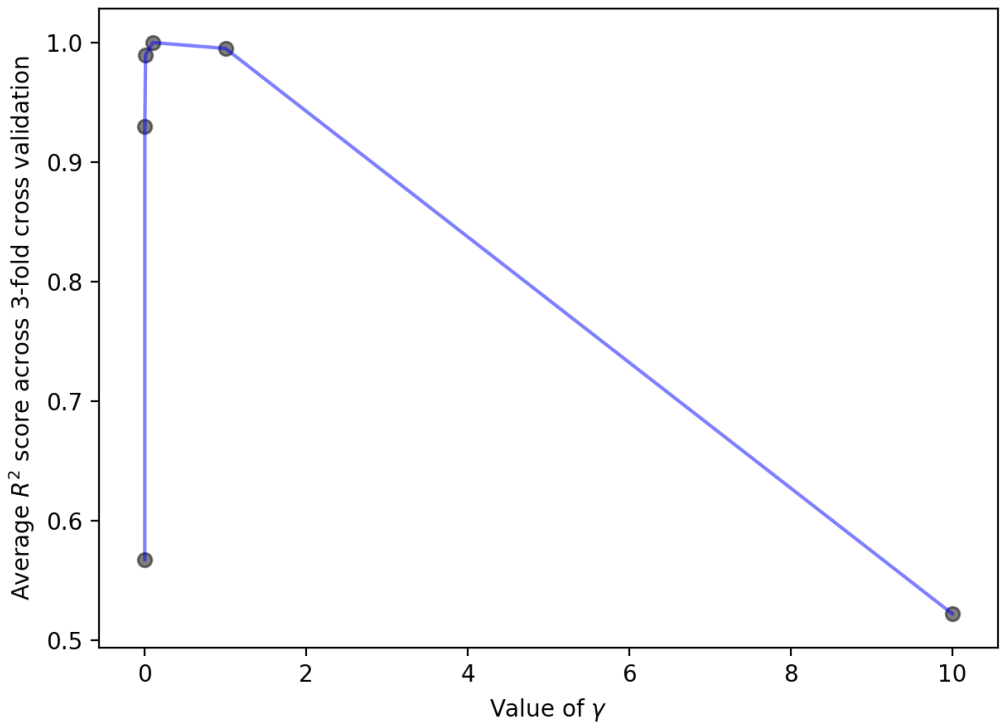
In [77]:

```
def plot_scores_wrt_gamma():
    %matplotlib notebook
    import matplotlib.pyplot as plt
    test_scores = answer_six()[1]
    gamma_range = np.logspace(-4,1,6)
    plt.figure(figsize =(7,5))
    plt.plot(gamma_range, test_scores, c = 'b', alpha = 0.5)
    plt.scatter(gamma_range, test_scores, c = 'k', alpha = 0.5)
    plt.xlabel('Value of $\gamma$')
    plt.ylabel('Average $R^2$ score across 3-fold cross validation')
    plt.show()

def answer_seven():

    # Your code here
    #plot_scores_wrt_gamma()
    # From the plot below, the 1st gamma value underfits the test set. (index 0)
    # The last gamma value overfits the training set. (index 5)
    # The 4th gamma value models the distribution the best. (index 3)
    gamma_range = np.logspace(-4,1,6)

    return (gamma_range[0], gamma_range[5], gamma_range[3]) # Return your answer
#answer_seven()
```



Out[77]:

```
(0.0001, 10.0, 0.10000000000000001)
```

In [ ]: