```
#
# ---

# # IU 4.6.5 Assignment
#
# In this assignment you'll explore the relationship
between model complexity and generalization
performance, by adjusting key parameters of various
supervised learning models.
This assignment will look at regression and
# ## Regression

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

# In[139]:
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

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
get_ipython().magic('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,
#part1_scatter()

# ### Question 1
#
# Write a function that fits a polynomial Linear
Regression 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.

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


# In[19]:  def
answer_one():
    from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import
PolynomialFeatures
```

```
    result = np.zeros((4,100))

    # paste Your code screenshot here




# use the function plot_one() to replicate the figure
# from the prompt once you have completed question one



# 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[26]:  def
answer_two():
    from sklearn.linear_model import LinearRegression
    from sklearn.preprocessing import
PolynomialFeatures
    from sklearn.metrics.regression import r2_score

 # Copy paste Your answer screenshot here
```

```
# ### 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?
- Note: there may be multiple correct solutions to this question.

```
#
```

```
# (Hint: Try plotting the $R^2$ scores from question 2
to visualize the relationship between degree level and
$R^2$)
#
# *This function should return one tuple with the
degree values in this order: `(Underfitting,
Overfitting, Good_Generalization)`*

# In[34]:  def
answer_three():
```

paste your coding screenshot

```
# ## 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?re
f=datanews.io) stored in `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
#
# <br>
#
# 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[4]:
import pandas as pd import numpy as np from
sklearn.model_selection import train_test_split

mush_df = pd.read_csv('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_state=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.*
#


# In[102]:
 def
answer_five():
    from sklearn.tree import DecisionTreeClassifier

    tree_clf = DecisionTreeClassifier().fit(X_train2,
y_train2)
    feature_names = []

    # Get index of importance leves since their's order
is the same with feature columns     for index,
importance in enumerate(tree_clf.feature_importances_):
        # Add importance so we can further order this
list, and add feature name with index
feature_names.append([importance,
X_train2.columns[index]])

    # Descending sort
    feature_names.sort(reverse=True)
    # Turn in to a numpy array
    feature_names = np.array(feature_names)
    # Select only feature names
feature_names = feature_names[:5,1]     #
Turn back to python list
    feature_names = feature_names.tolist()

    return feature_names # Your answer here

# In[103]:


answer_five()

# ### 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[131]:  def
answer_six():
    from sklearn.svm import SVC
from sklearn.model_selection import
validation_curve

    svc = SVC(kernel='rbf', C=1, random_state=0)
gamma = np.logspace(-4,1,6)     train_scores,
test_scores = validation_curve(svc, X_subset,
y_subset,
                            param_name='gamma',
param_range=gamma,
scoring='accuracy')

    scores = (train_scores.mean(axis=1),
test_scores.mean(axis=1))

    return scores # Your answer here
# In[137]:


print(answer_six())
 for index, num in enumerate(np.logspace(-
4,1,6)):
    print(num)

# ### 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)? Note: there may be multiple
correct solutions to this question.
#
# (Hint: Try plotting the scores from question 6 to
visualize the relationship between gamma and accuracy.)
#
# *This function should return one tuple with the
degree values in this order: `(Underfitting,
Overfitting, Good_Generalization)`*


# In[ ]:  def
answer_seven():

    # Your code here

    return (0.001, 10, 0.1)# Return your answer


# In[ ]:
```