A linear regressor using l2 regularization (sklearn.linear\_models.Ridge) was instantiated with default parameters, fit to training data and evaluated on a separate test set. Training score (r-squared) is 0.95 and test score is 0.72.

Answer the following questions and provide explanations:

1. Is the model over- or underfitting?

2. Should we reduce or increase complexity?

3. Should we try a higher or lower regularization coefficient alpha next?

Note that r-squared has a maximum value of 1.

1. The model is overfitting. The training score is high, while test score is low. It means the model performs well on data it has already seen, and thus overfits.

2. We should reduce the complexity of model. Reducing the complexity means model would be able to generalize better on unseen data.

3. Since we should be reducing complexity, hence we should try higher alpha. Higher alpha means more regularization, which results in less complexity.

A decision tree classifier (sklearn.tree.DecisionTreeClassifier) was instantiated with default parameters, fit to training data and evaluated on a separate test set. Training score (accuracy) is 0.92 and test score is 0.75.

Answer the following questions and provide explanations:

1. Is the model over- or underfitting?

2. Should we reduce or increase complexity?

3. Should we try a higher or lower number of maximum levels (max\_depth) next?

Note that accuracy has a maximum value of 1.0

1. The model is overfitting. The training score is high, while test score is low. It means the model performs well on data it has already seen, and thus overfits.

2. We should reduce the complexity of model. Reducing the complexity means model would be able to generalize better on unseen data.

3. Since we should be reducing complexity, hence we should try lower max\_depth. Lower max\_depth would mean the model would stop iterations sooner resulting in less complexity.

With random forest classifier imported using:

from sklearn.ensemble import RandomForestClassifier

instantiate a random forest classifier object with 50 trees, each tree having a maximum depth of 10, and a random state of 33, assigned to a variable model (all other parameters are default).

model = RandomForestClassifier(n\_estimators=50, max\_depth=10, random\_state=33)

With decision tree regressor imported using:

from sklearn.tree import DecisionTreeRegressor

instantiate a decision tree regressor object with a maximum depth of 5, and maximum features set to 'log2', assigned to a variable model (all other parameters are default).

model = DecisionTreeRegressor(max\_depth=5, max\_features='log2') # assuming log2 is a variable

With a feature matrix X and target vector y available, write Python code that uses scikit-learn to solve a classification task. More precisely, the code should:

- Import sklearn functions and classes needed

- Split X, and y into training and test set with default parameters

- Instantiate a logistic regression classifier with default parameters

- Train the classifier on the training data

- Print training and test score (accuracy)

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y)

model = LogisticRegression()

model.fit(X\_train, y\_train)

# predict the data on training set

y\_train\_predicted = model.predict(X\_train)

# get the training accuracy

acc\_train = accuracy\_score(y\_train, y\_train\_predicted)

# predict the data on validation set

y\_test\_predicted = model.predict(X\_test)

# get the validation accuracy

acc\_test = accuracy\_score(y\_test, y\_test\_predicted)

print("Training score = {}, Test score = {}".format(acc\_train, acc\_test))