In [1]:
```
import ssl
try:
    _create_unverified_https_context = ssl._create_unverified_context
except AttributeError:
    pass
else:
    ssl._create_default_https_context = _create_unverified_https_context
```

In [2]:
```
from sklearn import datasets
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import cross_validate
from sklearn.model_selection import GridSearchCV
```

In [3]:
```
cal = datasets.fetch_california_housing()
```

Task 1. Regression

a. Evaluate the performance of Linear Regression Model and Gradient Boosting Tree Regression Model on our dataset (use all data and all features) both with default parameters. Use cross-validation (k =5) to evaluate the performance with r2 scoring.

In [4]:
```
# Linear Regression Model
linear_reg = LinearRegression()
reg = linear_reg.fit(cal.data, cal.target)
print('Linear Regression Fit Score', reg.score(cal.data, cal.target))

# Cross validation
linear_regression_cv = cross_validate(linear_reg, cal.data, cal.target, cv=
print('Linear Regression Cross-Validation r2 scores', linear_regression_cv[
```

```
Linear Regression Fit Score 0.6062326851998049
Linear Regression Cross-Validation r2 scores [0.54866323 0.46820691 0.550
78434 0.53698703 0.66051406]
```

In [5]:
```
# Gradient Boosting Tree Regression Model
gbr = GradientBoostingRegressor()
gbr_reg = gbr.fit(cal.data, cal.target)
print('Gradient Boosting Tree Regression Fit Score', gbr_reg.score(cal.data

# Cross validation
gbr_cv = cross_validate(gbr, cal.data, cal.target, cv=5, scoring='r2')
print('Gradient Boosting Tree Regression Cross-Validation r2 scores', gbr_c
```

```
Gradient Boosting Tree Regression Fit Score 0.8033237500356992
Gradient Boosting Tree Regression Cross-Validation r2 scores [0.6025313
0.69877396 0.71802327 0.65021286 0.67973314]
```

b. For the Gradient Boosting Tree, test different combinations of meta-parameters by grid search.

Try to explore number of estimators, depth of the tree and learning rate. Do not try too many combinations of parameters as it can slow down the program significantly (use 4 to 10 combinations is enough in this assignment).

In [6]:
```
parameters = {'learning_rate': [0.1, 1], 'n_estimators': [50, 100], 'max_de
grid_search = GridSearchCV(gbr, parameters)
grid_search.fit(cal.data, cal.target)
```

Out[6]:
```
GridSearchCV(cv=None, error_score=nan,
             estimator=GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.
0,
                                                 criterion='friedman_ms
e',
                                                 init=None, learning_rate
=0.1,
                                                 loss='ls', max_depth=3,
                                                 max_features=None,
                                                 max_leaf_nodes=None,
                                                 min_impurity_decrease=0.
0,
                                                 min_impurity_split=None,
                                                 min_samples_leaf=1,
                                                 min_samples_split=2,
                                                 min_weight_fraction_leaf
=0.0,
                                                 n_estimators=100,
                                                 n_iter_no_change=None,
                                                 presort='deprecated',
                                                 random_state=None,
                                                 subsample=1.0, tol=0.000
1,
                                                 validation_fraction=0.1,
                                                 verbose=0, warm_start=Fa
lse),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0.1, 1], 'max_depth': [2, 5],
                         'n_estimators': [50, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fals
e,
             scoring=None, verbose=0)
```

In [7]:
```
parameters = grid_search.cv_results_['params']
for index in range(0, len(parameters)):
    print('Parameters:', parameters[index])
    test_scores = [grid_search.cv_results_['split0_test_score'][index], gri
                   grid_search.cv_results_['split2_test_score'][index], grid
                   grid_search.cv_results_['split4_test_score'][index]]
    print('Test Scores:', test_scores)
    print('Average score:', sum(test_scores) / len(test_scores), '\n')
```

```
Parameters: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 50}
Test Scores: [0.5091085808239103, 0.6047194586660538, 0.6336493568410193,
0.5109309996132647, 0.6116042765322647]
Average score: 0.5740025344953026

Parameters: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 100}
Test Scores: [0.5666410189218594, 0.6610157103217504, 0.6934111240143579,
0.6302300804653864, 0.6371192859526369]
Average score: 0.6376834439351982

Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 50}
Test Scores: [0.6217386777442604, 0.7001487016079269, 0.7373506111016822,
0.5929473720810241, 0.6793612927704773]
Average score: 0.6663093310610743

Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 100}
Test Scores: [0.623667069100114, 0.708855095326083, 0.7451923585909442,
0.44019395964779795, 0.7096038369805289]
Average score: 0.6455024639290936

Parameters: {'learning_rate': 1, 'max_depth': 2, 'n_estimators': 50}
Test Scores: [0.5141684560618235, 0.6790427937211255, 0.6821791304304752,
0.4739106557500233, 0.5816017424005754]
Average score: 0.5861805556728046

Parameters: {'learning_rate': 1, 'max_depth': 2, 'n_estimators': 100}
Test Scores: [0.4947976673229243, 0.6679283867796783, 0.6870157738562199,
0.5174300839297584, 0.5835019921925062]
Average score: 0.5901347808162175

Parameters: {'learning_rate': 1, 'max_depth': 5, 'n_estimators': 50}
Test Scores: [0.4191888426202711, 0.6074894515314417, 0.585151986705077,
0.05379995230679324, 0.5772293936506172]
Average score: 0.44857192536283996

Parameters: {'learning_rate': 1, 'max_depth': 5, 'n_estimators': 100}
Test Scores: [0.3830498185465445, 0.5939736365818591, 0.5675443679421561,
0.10540885248135456, 0.5557009927919687]
Average score: 0.4411355336687766
```

c. Briefly discuss the performance and summarize your findings.

From part a cross validation, we found that the linear regression model is a better fit than the gradient boosting tree regression model since the R^2 scores are lower, meaning the error is lower.

For part b, with the parameters tested, the combination of a learning rate of 0.1, depth of the tree of 5, and 50 estimators seemed to have the highest score, so a gradient boosting tree regression model with those parameters seems to be the best fit for this data.

Task 2. Classification

a. Evaluate the performance of Logistic Regression Model and Gradient Boosting Tree Classification Model on our dataset (use all data and all features) both with default parameters. Use cross-validation (k =5) to evaluate the performance with accuracy scoring.

In [22]:
```
X = cal.data
# set y to 1 if target > 2, 0 otherwise
y = 1 * (cal.target > 2)


lr = LogisticRegression()
# Cross validation
LRM_cv = cross_validate(lr, X, y, cv=5, scoring='accuracy')
print('Logistic Regression Scores', LRM_cv['test_score'])
```

```
/Users/leannahue/Library/Python/3.7/lib/python/site-packages/sklearn/line
ar_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://s
cikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression (https://scikit-learn.org/stable/modules/linear_model.html#logis
tic-regression)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/Users/leannahue/Library/Python/3.7/lib/python/site-packages/sklearn/line
ar_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge
 (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://s
cikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression (https://scikit-learn.org/stable/modules/linear_model.html#logis
tic-regression)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/Users/leannahue/Library/Python/3.7/lib/python/site-packages/sklearn/line
ar_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge
 (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://s
cikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression (https://scikit-learn.org/stable/modules/linear_model.html#logis
tic-regression)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)

Logistic Regression Scores [0.80208333 0.79457364 0.77664729 0.74006783
0.81782946]

/Users/leannahue/Library/Python/3.7/lib/python/site-packages/sklearn/line
```

```
ar_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://s
cikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression (https://scikit-learn.org/stable/modules/linear_model.html#logis
tic-regression)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/Users/leannahue/Library/Python/3.7/lib/python/site-packages/sklearn/line
ar_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge
 (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://s
cikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression (https://scikit-learn.org/stable/modules/linear_model.html#logis
tic-regression)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

In [24]:
```
X = cal.data
# set y to 1 if target > 2, 0 otherwise
y = 1 * (cal.target > 2)


gbc = GradientBoostingClassifier()
# Cross validation
GBC_cv = cross_validate(gbc, X, y, cv=5, scoring='accuracy')
print('Gradient Boosting Classifier Scores', GBC_cv['test_score'])
```

```
Gradient Boosting Classifier Scores [0.79093992 0.75436047 0.81177326 0.7
5678295 0.82897287]
```

b. For the Gradient Boosting Classification Tree, test different combinations of meta-parameters by grid search. Try to explore number of estimators, depth of the tree and learning rate. Do not try too many combinations as it can slow down the program significantly. (use 4 to 10 combinations is enough in this assignment)

In [18]:
```
parameters = {'learning_rate': [0.1, 1], 'n_estimators': [50, 100], 'max_de
grid_search_classifier = GridSearchCV(gbc, parameters)
grid_search_classifier.fit(X, y)
```

Out[18]:
```
GridSearchCV(cv=None, error_score=nan,
             estimator=GradientBoostingClassifier(ccp_alpha=0.0,
                                                  criterion='friedman_ms
e',
                                                  init=None, learning_rat
e=0.1,
                                                  loss='deviance', max_de
pth=3,
                                                  max_features=None,
                                                  max_leaf_nodes=None,
                                                  min_impurity_decrease=
0.0,
                                                  min_impurity_split=Non
e,
                                                  min_samples_leaf=1,
                                                  min_samples_split=2,
                                                  min_weight_fraction_lea
f=0.0,
                                                  n_estimators=100,
                                                  n_iter_no_change=None,
                                                  presort='deprecated',
                                                  random_state=None,
                                                  subsample=1.0, tol=0.00
01,
                                                  validation_fraction=0.
1,
                                                  verbose=0, warm_start=F
alse),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0.1, 1], 'max_depth': [2, 5],
                         'n_estimators': [50, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fals
e,
             scoring=None, verbose=0)
```

In [19]:
```
parameters = grid_search.cv_results_['params']
for index in range(0, len(parameters)):
    print('Parameters:', parameters[index])
    test_scores = [grid_search_classifier.cv_results_['split0_test_score'][
                   grid_search_classifier.cv_results_['split2_test_score'][i
                   grid_search_classifier.cv_results_['split4_test_score'][i
    print('Test Scores:', test_scores)
    print('Average score:', sum(test_scores) / len(test_scores), '\n')
```

```
Parameters: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 50}
Test Scores: [0.783187984496124, 0.8258236434108527, 0.814437984496124,
0.7308624031007752, 0.8345445736434108]
Average score: 0.7977713178294573

Parameters: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 100}
Test Scores: [0.8054748062015504, 0.7894864341085271, 0.8052325581395349,
0.7546027131782945, 0.8464147286821705]
Average score: 0.8002422480620155

Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 50}
Test Scores: [0.7625968992248062, 0.7204457364341085, 0.8158914728682171,
0.7381298449612403, 0.8323643410852714]
Average score: 0.7738856589147287

Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 100}
Test Scores: [0.7589631782945736, 0.6964631782945736, 0.8125, 0.725290697
6744186, 0.8253391472868217]
Average score: 0.7637112403100774

Parameters: {'learning_rate': 1, 'max_depth': 2, 'n_estimators': 50}
Test Scores: [0.782218992248062, 0.653827519379845, 0.8011143410852714,
0.7846414728682171, 0.8248546511627907]
Average score: 0.7693313953488372

Parameters: {'learning_rate': 1, 'max_depth': 2, 'n_estimators': 100}
Test Scores: [0.7834302325581395, 0.6618217054263565, 0.8040213178294574,
0.7572674418604651, 0.7926356589147286]
Average score: 0.7598352713178296

Parameters: {'learning_rate': 1, 'max_depth': 5, 'n_estimators': 50}
Test Scores: [0.7587209302325582, 0.6312984496124031, 0.781734496124031,
0.7449127906976745, 0.7662306201550387]
Average score: 0.7365794573643412

Parameters: {'learning_rate': 1, 'max_depth': 5, 'n_estimators': 100}
Test Scores: [0.75, 0.6063468992248062, 0.7679263565891473, 0.74321705426
35659, 0.7630813953488372]
Average score: 0.7261143410852713
```

c. Repeat the above (a and b) steps for using Area Under the Receiver Operating Characteristic Curve (ROC AUC) as the scoring.

In [20]:
```python
# a
# Logistic Regression
lr = LogisticRegression()
LRM_cv = cross_validate(lr, X, y, cv=5, scoring='roc_auc')
print('Logistic Regression ROC AUC Scores', LRM_cv['test_score'])

# Gradient Boosting Classifier
gbc = GradientBoostingClassifier()
# Cross validation
GBC_cv = cross_validate(gbc, X, y, cv=5, scoring='roc_auc')
print('Gradient Boosting Classifier ROC AUC Scores', LRM_cv['test_score'])
```

```
/Users/leannahue/Library/Python/3.7/lib/python/site-packages/sklearn/line
ar_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://s
cikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression (https://scikit-learn.org/stable/modules/linear_model.html#logis
tic-regression)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/Users/leannahue/Library/Python/3.7/lib/python/site-packages/sklearn/line
ar_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge
 (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://s
cikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression (https://scikit-learn.org/stable/modules/linear_model.html#logis
tic-regression)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/Users/leannahue/Library/Python/3.7/lib/python/site-packages/sklearn/line
ar_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge
 (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://s
cikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression (https://scikit-learn.org/stable/modules/linear_model.html#logis
tic-regression)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/Users/leannahue/Library/Python/3.7/lib/python/site-packages/sklearn/line
ar_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge
 (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://s
cikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression (https://scikit-learn.org/stable/modules/linear_model.html#logis
tic-regression)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/Users/leannahue/Library/Python/3.7/lib/python/site-packages/sklearn/line
ar_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge
 (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://s
cikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression (https://scikit-learn.org/stable/modules/linear_model.html#logis
tic-regression)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)

Logistic Regression ROC AUC Scores [0.87147593 0.88084579 0.86767746 0.81
723467 0.89979275]
Gradient Boosting Classifier Scores [0.87147593 0.88084579 0.86767746 0.8
1723467 0.89979275]
```

In [21]:
```
# b
parameters = {'learning_rate': [0.1, 1], 'n_estimators': [50, 100], 'max_de
grid_search_classifier = GridSearchCV(gbc, parameters, scoring='roc_auc')
grid_search_classifier.fit(X, y)

parameters = grid_search.cv_results_['params']
for index in range(0, len(parameters)):
    print('Parameters:', parameters[index])
    test_scores = [grid_search_classifier.cv_results_['split0_test_score'][
                   grid_search_classifier.cv_results_['split2_test_score'][i
                   grid_search_classifier.cv_results_['split4_test_score'][i
    print('Test Scores:', test_scores)
    print('Average score:', sum(test_scores) / len(test_scores), '\n')
```

```
Parameters: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 50}
Test Scores: [0.8579150719418577, 0.8924280124299515, 0.8987786821903123,
0.8761122573989296, 0.9117775370948371]
Average score: 0.8874023122111776

Parameters: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 100}
Test Scores: [0.8836412270968326, 0.8771819299915408, 0.9005757850761668,
0.8910138119127748, 0.9220270289455592]
Average score: 0.8948879566045747

Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 50}
Test Scores: [0.87553856547426, 0.8147136280041536, 0.908492348689206, 0.
8758924764917433, 0.9155469977022294]
Average score: 0.8780368032723185

Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 100}
Test Scores: [0.8703148292325207, 0.7769480022673403, 0.9110470761801285,
0.8568231445775827, 0.9100459674471643]
Average score: 0.8650358039409471

Parameters: {'learning_rate': 1, 'max_depth': 2, 'n_estimators': 50}
Test Scores: [0.8777959210973311, 0.7246752728074118, 0.9120219554750588,
0.885938400716772, 0.9059411896768996]
Average score: 0.8612745479546946

Parameters: {'learning_rate': 1, 'max_depth': 2, 'n_estimators': 100}
Test Scores: [0.8884326192879821, 0.7399952170285333, 0.9147633824558297,
0.8619518469295152, 0.8790574640012301]
Average score: 0.8568401059406181

Parameters: {'learning_rate': 1, 'max_depth': 5, 'n_estimators': 50}
Test Scores: [0.8472372324976445, 0.704262955645398, 0.8812800559713523,
0.8235354194916191, 0.8592596745678956]
Average score: 0.8231150676347818

Parameters: {'learning_rate': 1, 'max_depth': 5, 'n_estimators': 100}
Test Scores: [0.8392948052310502, 0.667943890066721, 0.865308828864896,
0.8329641046171554, 0.855783300651841]
Average score: 0.8122589858863327
```

d. Briefly discuss the performance and summarize your findings. Are they good classifiers?

Compare the result with a trivial classifier. Compare the results when using accuracy and ROC_AUC.

From part a with accuracy scores, the logistic regression model and the gradient boosting tree classification model seem to have similar results. For part b, with the parameters tested, the combination of a learning rate of 0.1, depth of the tree of 2, and 100 estimators seemed to have the highest score, so a gradient boosting tree classification model with those parameters seems to be the best fit for this data.

If we had a trivial classifier, both the accuracy and the ROC AUC score would be approximately 0.5. In comparison with a trivial classifier, the logistic regression model and the gradient boosting tree classification model seem to be good classifiers since accuracy and the ROC AUC scores of much over 0.5.

In [ ]: