STATS 202: DATA MINING AND ANALYSIS HOMEWORK #4

INSTRUCTOR: LINH TRAN, HOMEWORK #4, DUE DATE: AUGUST 11, 2023,
STANFORD UNIVERSITY, AND STUDENT: ADAM KAINIKARA

Note: For This HW I Would Like To Use Both of My Late Acceptance Free Passes

Part 1. **Problem 1 (10 points) Chapter 8, Exercise 4 (p. 362).**

See Picture

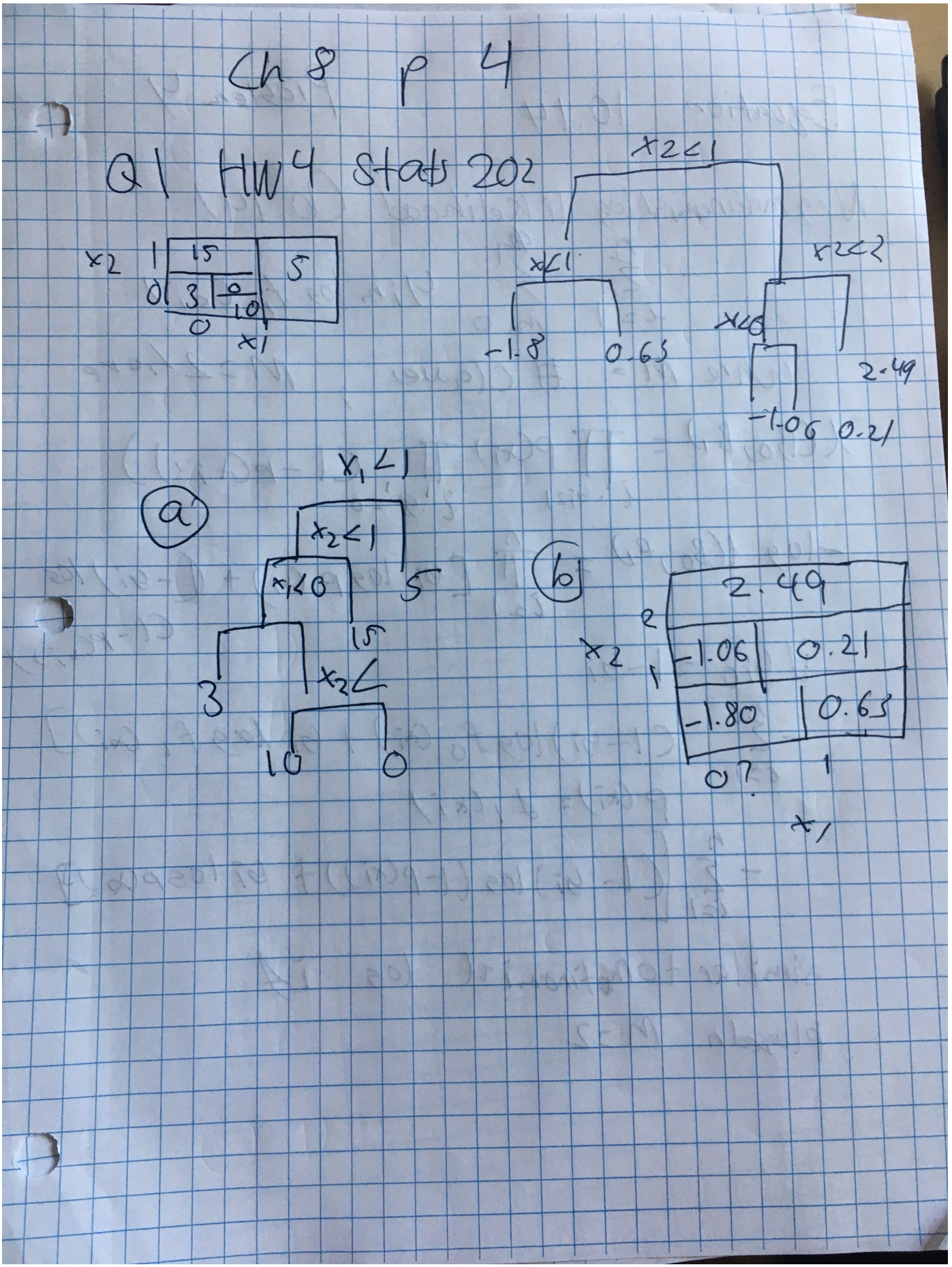


FIGURE 0.1. Partition Predictor Space, Tree Corresponding

Part 2. **Problem 2 (10 points) Chapter 8, Exercise 8 (p. 363).**

a) Loaded the data. Used the first 300 observations as training data and remaining as test data.

b)Below is the decision tree obtained. The text of the decisions is there but there is no way for me to zoom into it and make it visible without compromising viewing the entire tree.

Got training MSE of: training MSE [6.92555, 6.01548, 4.51463, 3.63198592, 2.780706106, 1.9490206, 1.3444905, 0.9582072, 0.6184801, 0.318548768, 0.1670528, 0.0808429, 0.034383, 0.008528, 0.0008045002, 1.5000000000000248e-06]. I used a max range of 1 to 100. After 1.5e-0.6 it becomes 0. As max range increases, training MSE decreased.

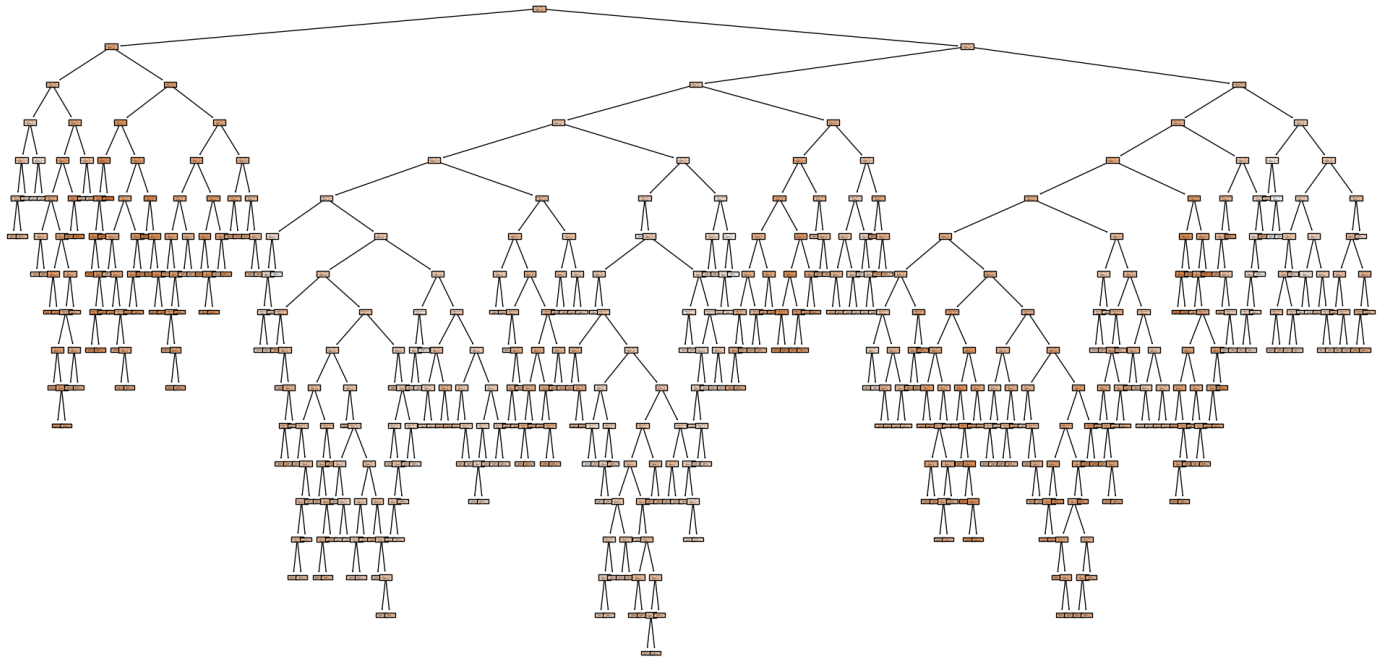


FIGURE 0.2. Decision Tree

c) Used 50 fold cross validation. Test MSE: [6.5926, 6.3392, 6.7435, 6.4871, 6.8642, 7.2016, 6.9819, 6.9257, 6.9609, 7.3169, 6.8377, 6.8694, 7.2913, 7.1097, 6.8807, 7.0151, 7.7205, 6.6586, 7.3512, 7.7440, 7.1296, 7.4179, 7.8666, 7.4043, 7.4034, 7.2810, 7.5770, 7.5597, 7.2096, 7.4301, 6.8801, 7.2097, 7.4453, 7.3694, 7.6687, 7.7158, 7.7056, 7.4671, 7.2502, 7.4823, 7.8073, 7.5877, 7.3345, 7.6565, 7.0715, 7.3730, 7.6791, 7.0353, 8.1583, 7.4158, 7.5074, 7.8412, 7.4064, 7.2881, 7.0706, 7.6808, 7.7743, 7.4709, 7.6717, 7.2558, 7.7232, 7.2517, 7.4788, 7.1370, 7.7501, 7.5733, 7.1611, 7.3242, 7.5635, 7.1397, 7.0876, 7.7972, 7.1674, 7.4782, 7.6583, 7.3527, 7.6408, 7.4755, 7.3450, 7.3133, 7.2930, 7.2004, 7.6946, 7.4337, 7.7125, 7.4135, 7.6839, 7.0104, 6.8647, 7.2573, 7.3605, 7.3134, 7.3692, 7.6356, 7.4716, 7.2165, 7.5625, 7.2234, 7.3998, 7.1178]

Cross-validation MSE: [7.7341, 7.4336, 6.6419, 7.4464, 7.6584, 8.2210, 8.3577, 8.5913, 8.4691, 8.4290, 8.9589, 9.0193, 8.9341, 9.3736, 9.0285, 9.3532, 8.9794, 8.8623, 9.0660, 8.8862, 9.4002, 9.3255, 9.2470, 9.3459, 9.7372, 9.0191, 9.0686, 9.1765, 8.8849, 9.2638, 9.1233, 9.0015, 9.3625, 9.1690, 8.8440, 9.2550, 9.4318, 8.9734, 9.2675, 9.1688, 9.6290, 9.7384, 8.9791, 9.4063, 9.1986, 8.8951, 9.1064, 9.0307, 8.6922, 8.8019, 9.2963, 9.4685, 9.4004, 9.3235, 9.1433, 8.9465,

8.8973, 9.3494, 8.9791, 8.9821, 9.0734, 8.9869, 8.6723, 9.2343, 9.6795, 9.0792, 9.0142, 9.1403, 9.2006, 9.1159, 9.1023, 8.9102, 9.2750, 9.2088, 8.8536, 8.8738, 8.8296, 9.1930, 9.5657, 9.2780, 9.6003, 9.5085, 9.0811, 9.1254, 9.7180, 9.0156, 9.1386, 9.0511, 9.2512, 9.0739, 9.4404, 9.3263, 9.1220, 9.2052, 9.1429, 9.3893, 9.1047, 9.1564, 9.1626, 9.2333]. In the long run Test MSE gradually increases. Here is a graphical representation of it.

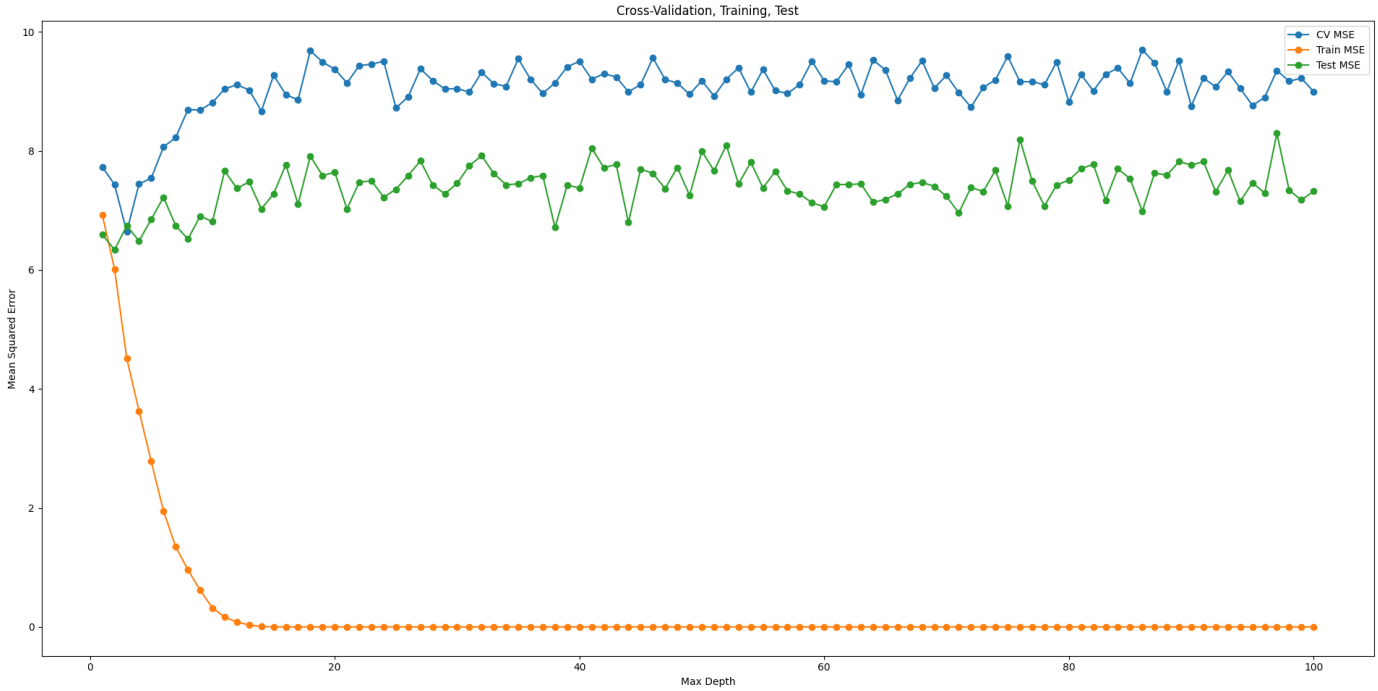


FIGURE 0.3. Cross Validation, Training, Test

d) Bagging reduced the error. CompPrice, Income, and Advertising where the 3 most important features. Got a bagging MSE of 4.8781.

e) Used a Random forest and varied max features.

Max Features: 0.2 Feature Importance: [0.13856 0.13733 0.12714 0.13383 0.23324 0.15025 0.07965]

Max Features: 0.4 Feature Importance: [0.14698 0.11921 0.12795 0.11191 0.28469 0.14311 0.06611]

Max Features: 0.6 Feature Importance: [0.17436 0.09798 0.13299 0.08696 0.32181 0.13276 0.05311]

Max Features: 0.8 Feature Importance: [0.18431 0.08943 0.13647 0.08047 0.32886 0.12985 0.05058]

In order this is CompPrice, Income, Advertising, Population, Price, ShelveLoc, Age, Education, Urban, US

Here is a graphical representation of how changing max features affected MSE.

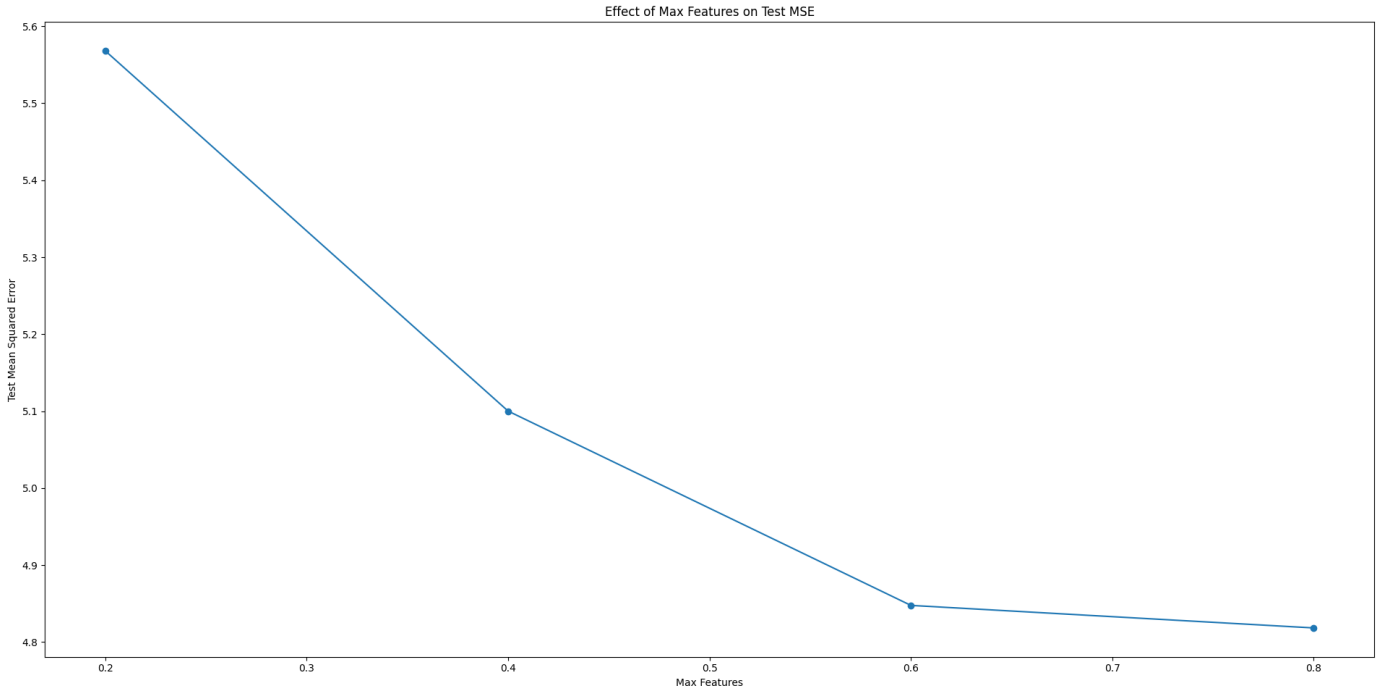


FIGURE 0.4. Random Forest

f) Not sure if BART completely worked. Went through a lot of loop holes for it to start. Got a MSE of 0.0479

Part 3. **Problem 3 (10 points) Chapter 8, Exercise 10 (p. 364).**

a) Imported the data. Made a loop that went over the salary column. If 'NA' was in the column, that individual was removed. In total about 63 people were removed. The remaining people had their salaries log transformed.

b) Split the data where training was first 200 observations, test was remaining. X was all the quantitative values. Y was log transformed salary.

c) Performed gradient boosting with 1000 trees. The range of values for λ where [0.001, 0.005,0.01, 0.05, 0.1, 0.15, 0.5, 1]. Below is a plot of different shrinkage values on the x-axis and the corresponding training set MSE on the y-axis.

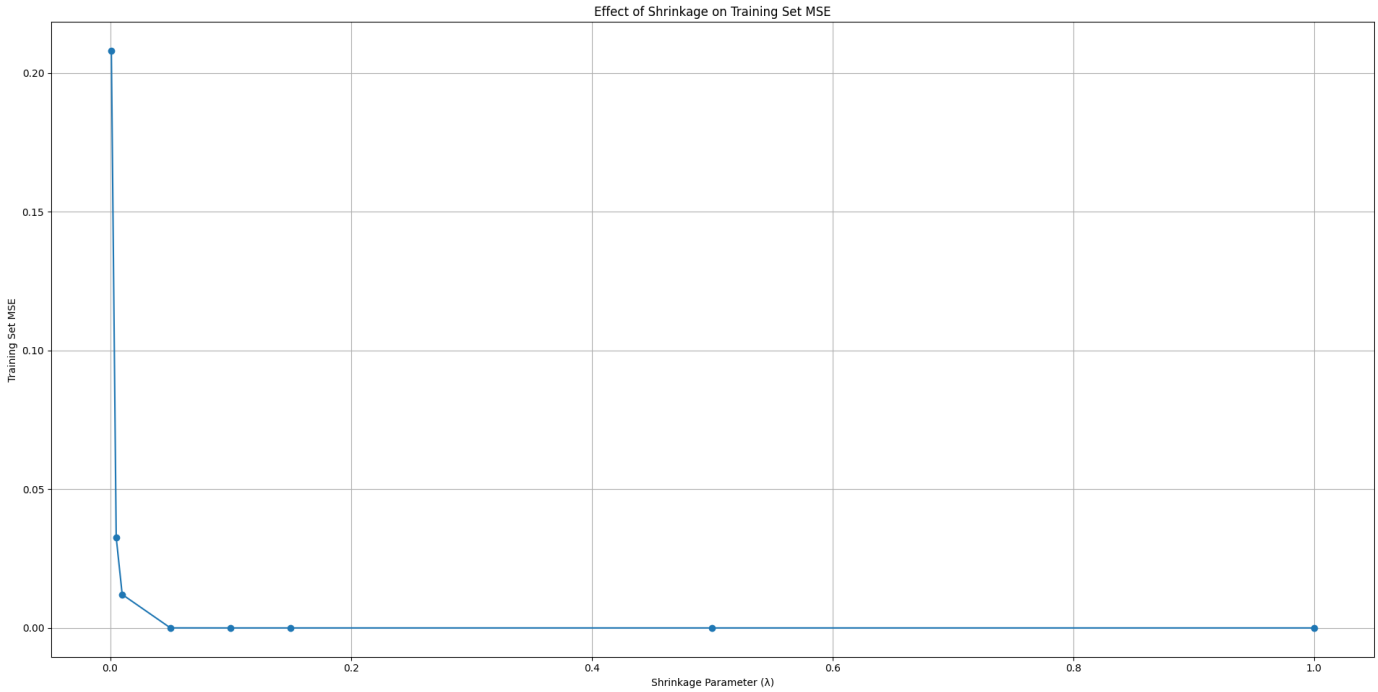


FIGURE 0.5. Shrinkage on Training

d) Same as above but now with test MSE.

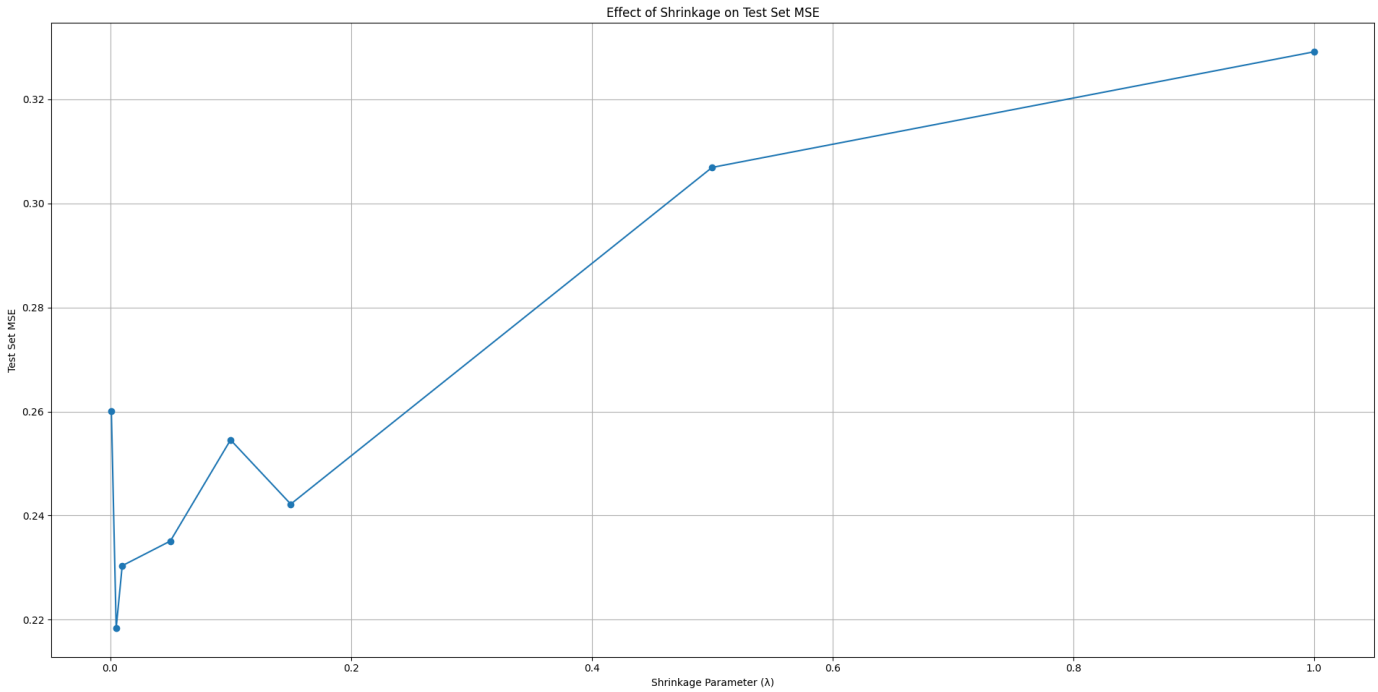


FIGURE 0.6. Shrinkage on Test

e) Train MSE Boosting [0.2079, 0.0326, 0.0120, 1.9360e-05, 1.5571e-08, 6.4497e-12, 2.2169e-16, 2.1421e-16]

Test MSE Boosting [0.2600, 0.2184, 0.2304, 0.2351, 0.2546, 0.2422, 0.3069, 0.3291]

The two regression approaches I used where Linear regression and Ridge regression.

Test MSE Linear 0.51394

Test MSE Ridge 0.5139, 0.5139, 0.5139, 0.5139, 0.5139, 0.5139, 0.5139, 0.5138, 0.5136

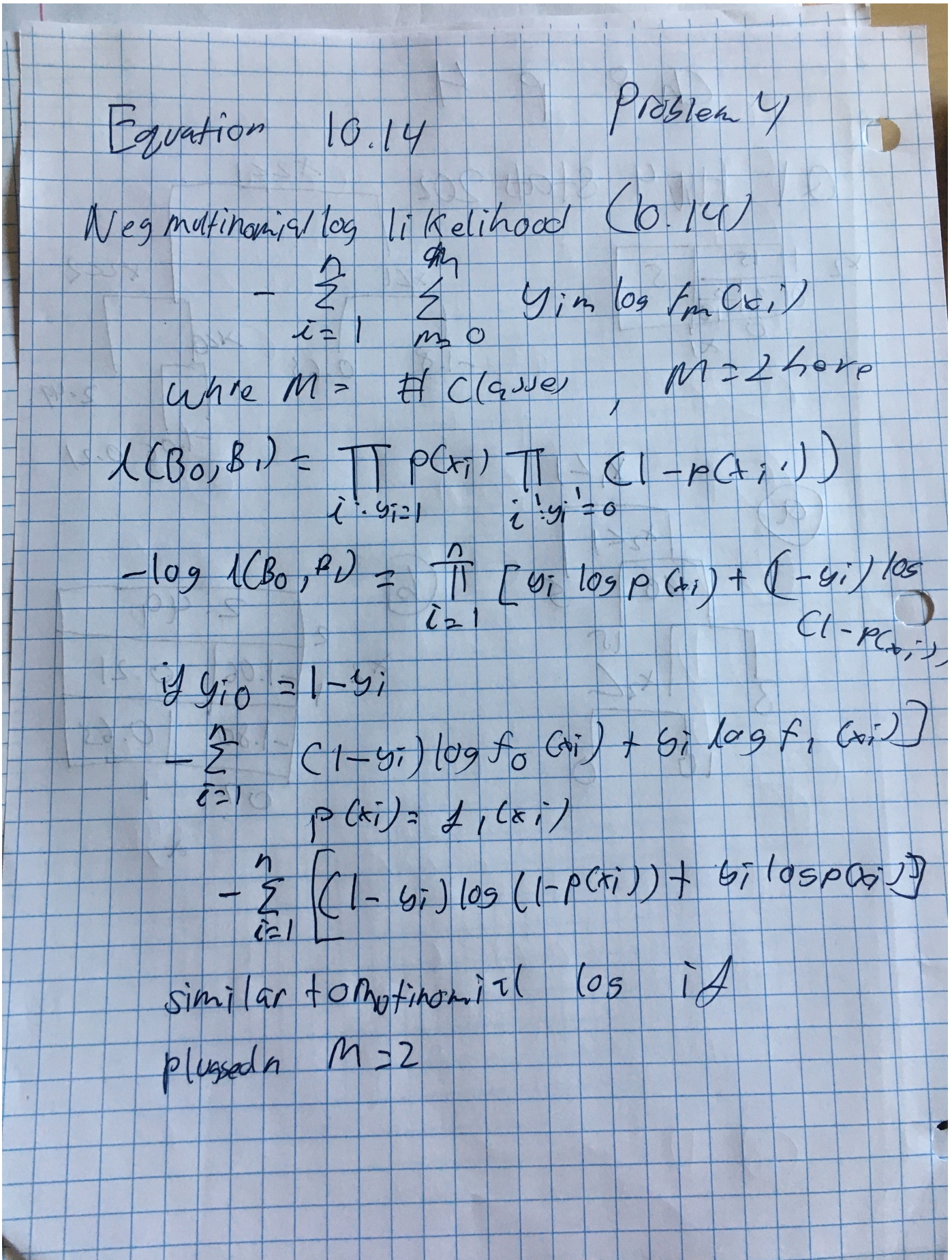
The test MSE of linear and ridge regressions where greater than that of boosting

f) Did feature importance. These where the values. The 3 most important variables where Catbat, Chits, Cruns.

('catbat', 0.5562) ('chits', 0.0875) ('cruns', 0.0525) ('walks', 0.0481) ('cwalks', 0.0419) ('chmrun', 0.0403) ('atbat', 0.0385) ('crbi', 0.0336) ('hits', 0.0279) ('years', 0.0241) ('putouts', 0.0160) ('rbi', 0.0132) ('assits', 0.0072) ('errors', 0.0070) ('hmrun', 0.0060)

g) Test MSE Bagging 0.22657

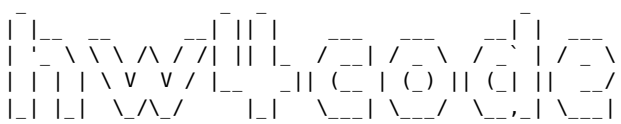
Part 4. **Problem 4 (10 points) Chapter 10, Exercise 3 (p. 459).**



Part 5. Problem 5 (Bonus 10 points) Let xi : i = 1, ..., p be the input predictor values and a(2s) k : k = 1, ..., K be the K-dimensional output from a 2-layer and M-hidden unit neural network with sigmoid activation sv(a) = {1 + e—a}—1 such that a(1s) j = w(1s) j0 + p∑ i=1 w(1s) ji xi : j = 1, ..., M a(2s) k = w(2s) k0 + M∑ j=1 w(2s) kj sv (a(1s) j) Show that there exists an equivalent network that computes exactly the same output values, but with hidden unit activation functions given by tanh(a) = ea—e—a ea +e—a , i.e. a(1t) j = w(1t) j0 + p∑ i=1 w(1t) ji xi : j = 1, ..., M a(2t) k = w(2t) k0 + M∑ j=1 w(2t) kj tanh (a(1t) j) Hint: first derive the relation between sv(a) and tanh(a). Then show that the parameters of the two networks differ by linear transformations.

Given:
$\sigma(a) = \frac{1}{1+e^{-a}}$
$tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$
Want to find the relationship between $\sigma(a)$ and $tanh(a)$
Factor out e^a as a common factor
$tanh(a) = \frac{e^a(1-e^{-2a})}{e^a(1+e^{-2a})} = \frac{1-e^{-2a}}{1+e^{-2a}} = \frac{1}{1+e^{-2a}} - \frac{e^{-2a}}{1+e^{-2a}}$
$\sigma(2a) = \frac{1}{1+e^{-2a}}$
Therefore
$tanh(a) = \sigma(2a) - \frac{e^{-2a}}{1+e^{-2a}}$
Now dealing with $\sigma(a) = \frac{1}{1+e^{-a}}$
$1 - \sigma(a) = 1 - \frac{1}{1+e^{-a}}$
$1 - \sigma(a) = \frac{1+e^{-a}}{1+e^{-a}} - \frac{1}{1+e^{-a}}$
$1 - \sigma(a) = \frac{e^{-a}}{1+e^{-a}}$
$1 - \sigma(2a) = \frac{e^{-2a}}{1+e^{-2a}}$
Therefore
$tanh(a) = \sigma(2a) - (1 - \sigma(2a))$
$tanh(a) = 2\sigma(2a) - 1$
$\sigma(2a) = \frac{tanh(a)-1}{2}$
Show that the parameters of the two networks differ by linear transformations
$a_k^{(2s)} = w_{k0}^{(2s)} + \Sigma_{j=1}^{M} w_{kj}^{(2s)} \sigma(a_j^{(1s)})$
$a_k^{(2t)} = w_{k0}^{(2t)} + \Sigma_{j=1}^{M} w_{kj}^{(2t)} tanh(a_j^{(1t)})$
Replace $tanh$ and σ with relationships found earlier
$a_k^{(2s)} = w_{k0}^{(2s)} + \Sigma_{j=1}^{M} w_{kj}^{(2s)} 2\frac{tanh(a)-1}{2}(a_j^{(1s)})$
Multiply by 2 cause it is $2a$
$a_k^{(2t)} = w_{k0}^{(2t)} + \Sigma_{j=1}^{M} w_{kj}^{(2t)} 2\sigma(2a-1)(a_j^{(1t)})$
$a_k^{(2s)} = w_{k0}^{(2s)} + \Sigma_{j=1}^{M} w_{kj}^{(2s)} tanh(a) - 1(a_j^{(1s)})$
If this was multiplied out and simplified it would become
$w_{kj}^{(2t)} = 2w_{kj}^{2s}$
$w_{k0}^{(2t)} = w_{k0}^{(2s)} - \Sigma w_{kj}^{2s}$

A linear transform can be seen between the two with the 2 and the constant



ch8p10.py

```
from numpy import *
import matplotlib.pyplot as plt
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
from sklearn import tree
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.ensemble import BaggingRegressor
#AtBat  Hits    HmRun   Runs    RBI     Walks   Years   CAtBat  CHits   CHmRun  CRuns   CRBI    CWalks  League  Division        PutOuts Assists Errors  Salary  NewLeague

hitters dt = dtype([('atbat', float64),('hits', float64),('hmrun', float64), ('runs', float64),('rbi', float64),('walks', float64),('years', float64),('catbat', float64)
,('chits', float64),('chmrun', float64),('cruns', float64),('crbi', float64),('cwalks', float64),('league','U10'),('division','U10'),('putouts', float64)
,('assits', float64),('errors', float64),('salary', object),('newleague', 'U10')])


def data_loader(fname):
    cata_data_a = loadtxt(fname,skiprows=1, usecols=(14,15,20), delimiter=',', dtype=str )
    quant_data_a = loadtxt(fname, skiprows=1, usecols=(1,2,3,4,5,6,7,8,9,10,11,12,13,16,17,18), delimiter=',', dtype=float64)
    mix_data_a = loadtxt(fname, skiprows=1, usecols=(19), delimiter=',', dtype=object)
    unfil_data_a = empty(quant_data_a.shape[0], dtype=hitters_dt)

    unfil_data_a['atbat'] = quant_data_a[:,0]
    unfil_data_a['hits'] = quant_data_a[:,1]
    unfil_data_a['hmrun'] = quant_data_a[:,2]
    unfil_data_a['runs'] = quant_data_a[:,3]
    unfil_data_a['rbi'] = quant_data_a[:,4]
    unfil_data_a['walks'] = quant_data_a[:,5]
    unfil_data_a['years'] = quant_data_a[:,6]
    unfil_data_a['catbat'] = quant_data_a[:,7]
    unfil_data_a['chits'] = quant_data_a[:,8]
    unfil_data_a['chmrun'] = quant_data_a[:,9]
    unfil_data_a['cruns'] = quant_data_a[:,10]
    unfil_data_a['crbi'] = quant_data_a[:,11]
    unfil_data_a['cwalks'] = quant_data_a[:,12]
    #unfil_data_a['league'] = cata_data_a[:,0]
    #unfil_data_a['division'] = cata_data_a[:,1]
    unfil_data_a['putouts'] = quant_data_a[:,13]
    unfil_data_a['assits'] = quant_data_a[:,14]
    unfil_data_a['errors'] = quant_data_a[:,15]
    unfil_data_a['salary'] = mix_data_a
    #unfil_data_a['newleague'] = cata_data_a[:,2]
    print(unfil_data_a.shape[0])
    return unfil_data_a

def filter_salary(unfil_data_a):
    deleted_l = []
    name_l = ['atbat', 'hits', 'hmrun', 'rbi', 'walks', 'years', 'catbat', 'chits', 'chmrun', 'cruns', 'crbi', 'cwalks', 'passouts', 'assits', 'errors', 'salary']

    for i in range(unfil_data_a.shape[0]):
        salary_v = unfil_data_a[i]['salary']
        if 'NA' in salary_v:
            deleted_l.append(i)
    print("deleted ppl", deleted_l)

    data_a = delete(unfil_data_a, deleted_l, axis=0)

    for i in range(data_a.shape[0]):
        salary_v = data_a[i]['salary']
        data_a[i]['salary'] = log(float(salary_v))
    #print("new list of pplwith salary log change",filtere_data_a)

    #salary_data = data_a['salary']

    return data_a

def make_x_y(data_a):
    name_l = ['atbat', 'hits', 'hmrun', 'rbi', 'walks', 'years', 'catbat', 'chits', 'chmrun', 'cruns', 'crbi', 'cwalks', 'putouts', 'assits', 'errors']
    #'hmrun', 'rbi', 'walks', 'years', 'catbat', 'chits', 'chmrun', 'cruns', 'crbi', 'cwalks', 'passouts', 'assits', 'errors']
    x_a = empty((data_a.shape[0], len(name_l)))
    for i, name  in  enumerate(name_l):
        x_a[:,i] = data_a[name]
    y_v = data_a['salary']

    return x_a, y_v

def des_tree(xtrain_a,ytrain_v,ytest_v):
    clf = tree.DecisionTreeClassifier()
    clf = clf.fit(xtrain_a, ytrain_v)
    clf.predict(ytrain_v)


def train_grad_boost(xtrain_a, ytrain_v):
    #def grad_boost(xtrain_a, ytrain_v, xtest_a, ytest_v):

    lamda_shrinkage = [0.001, 0.005,0.01, 0.05, 0.1, 0.15, 0.5, 1]
    train_error_l =[]
    for i in lamda_shrinkage:
        model = GradientBoostingRegressor(n_estimators=1000, learning_rate=i)
        model.fit(xtrain_a, ytrain_v)
        ypred_v = model.predict(xtrain_a)
        mse = mean_squared_error(ytrain_v, ypred_v)
        train_error_l.append(mse)
    print("train mse boost", train_error_l)
    plt.plot(lamda shrinkage, train error_l, marker='o')
    plt.xlabel('Shrinkage Parameter (λ)')
    plt.ylabel('Training Set MSE')
    plt.title('Effect of Shrinkage on Training Set MSE')
    plt.grid(True)
    plt.show()

def test_grad_boost(xtrain_a, ytrain_v, xtest_a, ytest_v):
    #def grad_boost(xtrain_a, ytrain_v, xtest_a, ytest_v):

    lamda_shrinkage = [0.001, 0.005,0.01, 0.05, 0.1, 0.15, 0.5, 1]
    test_error_l =[]
    for i in lamda_shrinkage:
        model = GradientBoostingRegressor(n_estimators=1000, learning_rate=i)
        model.fit(xtrain_a, ytrain_v)
        ytestpred_v = model.predict(xtest_a)
        mse = mean_squared_error(ytest_v, ytestpred_v)
        feature_importances = model.feature_importances_
        test_error_l.append(mse)
    print("test mse boost", test_error_l)
    plt.plot(lamda_shrinkage, test_error_l, marker='o')
    plt.xlabel('Shrinkage Parameter (λ)')
    plt.ylabel('Training Set MSE')
```

```
    plt.title('Effect of Shrinkage on Test Set MSE')
    plt.grid(True)
    plt.show()

    plt.show()

def find_imp_pred(xtrain_a, ytrain_v, name_l):
    model = GradientBoostingRegressor(n_estimators=1000, learning_rate=0.227625)
    #use this as learning rate cause it is the average of what was given earlier
    model.fit(xtrain_a, ytrain_v)
    feat_imp = model.feature_importances_
    feat_imp_d = {f: imp for f, imp in zip(name_l, feat_imp)}
    sort_feat = sorted(feat_imp_d.items(), key=lambda x: x[1], reverse=True)
    print(sort_feat)
    return sort_feat


def lin_reg(xtrain_a, ytrain_v, xtest_a, ytest_v):
    lin_model = LinearRegression()
    lin_model.fit(xtrain_a, ytrain_v)
    y_pred_lin = lin_model.predict(xtest_a)
    test_mse_linear = mean_squared_error(ytest_v, y_pred_lin)
    print("test_mse_linear", test_mse_linear)

def rid_reg(xtrain_a, ytrain_v, xtest_a, ytest_v):
    #like in grad boost and using different lamda values i will use dif alpha values for ridge
    alpha= [0.001, 0.005,0.01, 0.05,0.1,0.5, 1, 5, 10]
    rid_error = []
    for i in alpha:
        rid_model = Ridge(alpha=i)
        rid_model.fit(xtrain_a, ytrain_v)
        y_pred_rid = rid_model.predict(xtest_a)
        rid_error.append(mean_squared_error(ytest_v, y_pred_rid))

    print("test mse ridge", rid_error)

def bag(xtrain_a, ytrain_v, xtest_a, ytest_v):
    model = BaggingRegressor(n_estimators=1000, random_state=1)
    model.fit(xtrain_a, ytrain_v)
    y_pred_test = model.predict(xtest_a)
    mse_bag = mean_squared_error(ytest_v, y_pred_test)
    print("test mse bagging", mse_bag)
    return mse_bag
def main():
    unfdata_a = data_loader("Hitters.csv")
    data_a = filter_salary(unfdata_a)
    x_a, y_v = make_x_y(data_a)
    xtrain_a = x_a[:200]
    ytrain_v = y_v[:200]
    xtest_a = x_a[200:]
    ytest_v = y_v[200:]
    train_grad_boost(xtrain_a,ytrain_v)
    test_grad_boost(xtrain_a, ytrain_v, xtest_a,ytest_v)
    lin_reg(xtrain_a, ytrain_v, xtest_a,ytest_v)
    rid_reg(xtrain_a, ytrain_v, xtest_a,ytest_v)
    name_l = ['atbat', 'hits', 'hmrun', 'rbi', 'walks', 'years', 'catbat', 'chits', 'chmrun', 'cruns', 'crbi', 'cwalks', 'putouts', 'assits', 'errors']
    find_imp_pred(xtrain_a, ytrain_v, name_l)
    bag(xtrain_a, ytrain_v, xtest_a,ytest_v)
    raise SystemExit
    xtrain_a = filtered_log_a['']
    ytrain_v = filtered_log_a[:200]['salary']
    #xtest_a = filtered_log_a[200:][xtrain_dtype]
    #ytest_v = filtered_log_a[200:]['salary']
if __name__ == '__main__':
    main()
```

ch8p8.py

```python
from numpy import *
import matplotlib.pyplot as plt
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
from sklearn import tree
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.ensemble import BaggingRegressor

from sklearn.tree import DecisionTreeRegressor,plot_tree
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_predict
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import RandomForestRegressor

def data_loader(fname):
    x_a = loadtxt(fname, skiprows=1, usecols=(1,2,3,4,5,7,8), delimiter=',', dtype=float64)
    y_v = loadtxt(fname, skiprows=1, usecols=0, delimiter=',', dtype=float64)

    return x_a, y_v


def tree_things(train_x_a, train_y_v):
    tree_reg = DecisionTreeRegressor(max_depth=None)
    tree_reg.fit(train_x_a, train_y_v)
    ytrain_pred = tree_reg.predict(train_x_a)
    train_mse = mean_squared_error(train_y_v, ytrain_pred)
    plt.figure(figsize=(20, 15))
    plot_tree(tree_reg, filled=True)
    plt.show()

    return tree_reg, train_mse

def crossval_for_tree(xtrain, ytrain, xtest, ytest, maxrange):
    train_mse_l = []
    test_mse_l = []
    cross_val_l = []
    for i in maxrange:
        tree_reg = DecisionTreeRegressor(max_depth=i)
        ypred = cross_val_predict(tree_reg, xtrain, ytrain, cv=50)
        cross_val_mse = mean_squared_error(ytrain, ypred)
        cross_val_l.append(cross_val_mse)

        tree_reg.fit(xtrain, ytrain)
        ytrainpred = tree_reg.predict(xtrain)
        trainmse = mean_squared_error(ytrain, ytrainpred)
        train_mse_l.append(trainmse)

        ytestpred = tree_reg.predict(xtest)
        testmse = mean_squared_error(ytest, ytestpred)
        test_mse_l.append(testmse)
    print("training mse", train_mse_l)
    print("test mse", test_mse_l)
    print("cross val mse", cross_val_l)
    plt.figure(figsize=(20, 15))
    plt.plot(maxrange, cross_val_l, marker='o', label='CV MSE')
    plt.plot(maxrange, train_mse_l, marker='o', label='Train MSE')
    plt.plot(maxrange, test_mse_l, marker='o', label='Test MSE')
    plt.xlabel('Max Depth')
    plt.ylabel('Mean Squared Error')
    plt.title('Cross-Validation, Training, Test')
    plt.legend()
    plt.show()

    return train_mse_l, test_mse_l, cross_val_l

def doin_bagging(xtrain, ytrain, xtest, ytest):
    ''' so basicaly do a tree then bagging?'''
    tree_reg = DecisionTreeRegressor()
    bag_reg = BaggingRegressor(base_estimator=tree_reg, n_estimators=1000)
    bag_reg.fit(xtrain, ytrain)
    ytestpred = bag_reg.predict(xtest)
    testmse = mean_squared_error(ytest, ytestpred)
    print("bagg mse", testmse)
    #test_mse, feature_importances = bagging_regression(xtrain, ytrain, xtest, ytest)

def forest_stuff(xtrain, ytrain, xtest, ytest, n_estimators=1000, max_features_values=None):
    teast_mse_l = []
    feature_importances = []

    for max_features in max_features_values:
        rf_reg = RandomForestRegressor(n_estimators=n_estimators, max_features=max_features, random_state=42)
        rf_reg.fit(xtrain, ytrain)
        y_test_pred = rf_reg.predict(xtest)
        test_mse = mean_squared_error(ytest, y_test_pred)
        teast_mse_l.append(test_mse)
        feature_importances.append(rf_reg.feature_importances_)


```

```
    plt.figure(figsize=(10, 6))
    plt.plot(max_features_values, teast_mse_l, marker='o')
    plt.xlabel('Max Features')
    plt.ylabel('Test Mean Squared Error')
    plt.title('Effect of Max Features on Test MSE')
    plt.show()
    for idx, max_features in enumerate(max_features_values):
        print(f"Max Features: {max_features}")
        print("Feature Importances:", feature_importances[idx])
        print()
    return teast_mse_l, feature_importances



def bart(train_x, train_y, test_x, test_y, num_trees=100, num_burn_in=100, num_iterations=1000):
    pass
    ''' note: never really used it before: heavily influenced from online tutotrials such as https://allenai.github.io/pybart/'''
    train_x = train_x.astype(float32)
    train_y = train_y.astype(float32)
    test_x = test_x.astype(float32)
    test_y = test_y.astype(float32)
    model = Model()
    model.num_trees = num_trees
    model.num_burn_in = num_burn_in
    model.num_iterations = num_iterations

    model.fit(train_x, train_y)

    y_test_pred = model.predict(test_x)

    test_mse = np.mean((test_y - y_test_pred) ** 2)
    print("Bart Test MSE:", test_mse)

    return test_mse



def main():
    x_a, y_v = data_loader("Carseats.csv")

    train_x_a = x_a[:300]
    train_y_v = y_v[:300]
    test_x_a = x_a[300:]
    test_y_v = y_v[300:]

#turn this back on later for decs tree pic
    tree_things(train_x_a, train_y_v)


    maxrange = range(1, 101)
    crossval_for_tree(train_x_a,train_y_v,test_x_a,test_y_v,maxrange)
    doin_bagging(train_x_a,train_y_v,test_x_a,test_y_v)
    max_features_values = [None, 0.2, 0.4, 0.6, 0.8]
    teast_mse_, feature_importances = forest_stuff(train_x_a, train_y_v, test_x_a, test_y_v, max_features_values=max_features_values)
    #test_mse = bart(train_x_a, train_y_v, test_x_a, test_y_v)


if __name__ == '__main__':
    main()
```