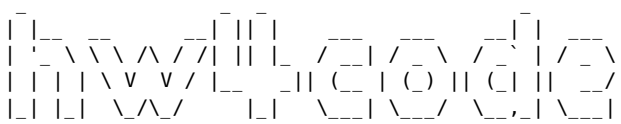

ch8p10.py

```
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()
```