Homework 4: Moving Beyond Linearity

Tianyue Niu

In [517]:
```
#import necessary packages
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import cross_val_score
from patsy import dmatrix
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import ElasticNet
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression
from sklearn.inspection import plot_partial_dependence
from mlxtend.evaluate import feature_importance_permutation
```

In [518]:
```
#read in data
test = pd.read_csv('gss_test.csv')
train = pd.read_csv('gss_train.csv')
```

Egalitarianism and income

1)*(20 points) Perform polynomial regression to predict egalit_scale as a function of income06. Use and plot 10-fold cross-validation to select the optimal degree d for the polynomial based on the MSE. Plot the resulting polynomial fit to the data, and also graph the average marginal effect (AME) of income06 across its potential values. Be sure to provide substantive interpretation of the results.*

In [519]:
```
#Didn't use sklearn PolynomialFeatures because the function automatically includes inter
action
#terms in the regression fit and these interaction terms can't be removed.
```

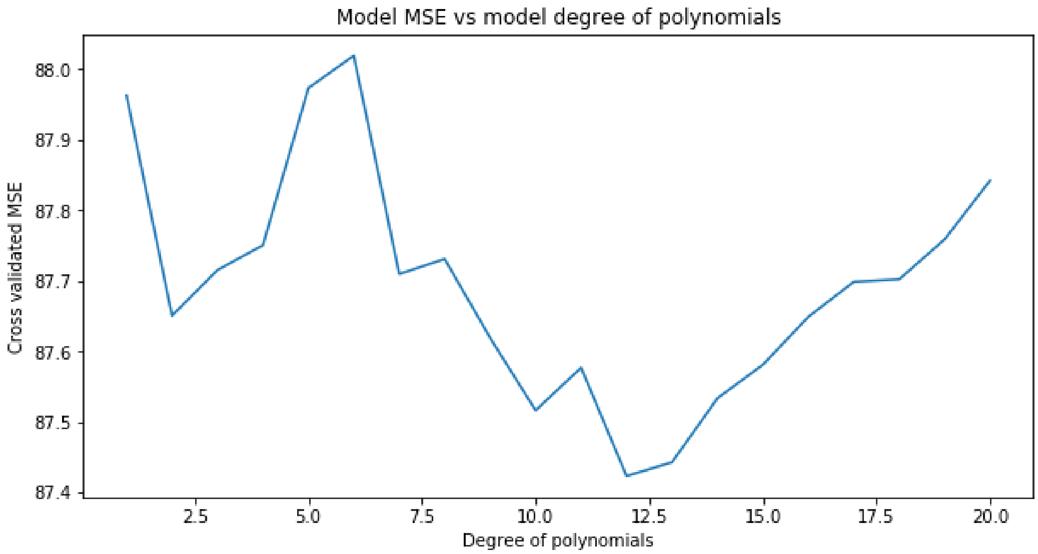
In [520]:
```
#define X and y
X = np.array(train['income06']).reshape(-1,1)
y = np.array(train['egalit_scale']).reshape(-1,1)
```

In [521]:
```
def find_best_deg(X, y):
    cv = {}
    min_error = 10000000
    min_deg = 0
    degrees = np.arange(1, 21)
    x = pd.DataFrame()

    for deg in degrees:
        #manually add polynomials of X to x
        x[deg] = X ** deg
        # Fit polynomial regression
        lr = LinearRegression()
        #get error
        error = np.sum(-cross_val_score(lr, x, y, cv=10, scoring='neg_mean_squared_erro
r'))/10
        cv[deg] = error
        if error < min_error:
            min_error = error
            min_deg = deg

    return min_deg, cv
```

In [522]:
```
find_best_deg(train['income06'], y)
```

Out[522]:
```
(12,
 {1: 87.96300193115194,
  2: 87.65085217286715,
  3: 87.7154238094978,
  4: 87.7506412125094,
  5: 87.97322944676486,
  6: 88.01923052537474,
  7: 87.70981059410886,
  8: 87.73114625218366,
  9: 87.61920345310567,
  10: 87.51660188653618,
  11: 87.5770963856581,
  12: 87.42373138398571,
  13: 87.44309717782467,
  14: 87.53385243089564,
  15: 87.58130024172851,
  16: 87.64923123298351,
  17: 87.69882301886456,
  18: 87.70242260890136,
  19: 87.75905670902408,
  20: 87.84239432606995})
```

In [523]:
```
#Plot the MSE graph
lists = sorted(find_best_deg(train['income06'],y)[1].items())
degree, mse = zip(*lists)
plt.figure(figsize=(10,5))
plt.plot(degree, mse)
plt.xlabel('Degree of polynomials')
plt.ylabel('Cross validated MSE')
plt.title('Model MSE vs model degree of polynomials');
```



We see that the MSE fluctuates as degree of polynomials increases. The cross-validated MSE is lowest at degree = 12. There also exists a local minimum at degree = 2.

In [524]:
```
def calculate_ame (X, y, degree):

    x = pd.DataFrame()
    x_drv = pd.DataFrame()
    ame_total = []

    for deg in range(degree+1):
        if deg != 0:
            x[deg] = X ** deg
    lm = LinearRegression().fit(x, y)
    coefs = lm.coef_

    for deg in range(degree):
        x_drv[deg] = (deg+1)* coef[deg] * (X ** deg)
        ame_x = x_drv[deg].mean()
        ame_total.append(ame_x)
    ame = np.sum(ame_total)

    return ame, coefs
```

In [525]:
```
ame, coef_degree_12 = calculate_ame(train['income06'], train['egalit_scale'], 12)
```

In [526]:
```
coef_degree_12
```

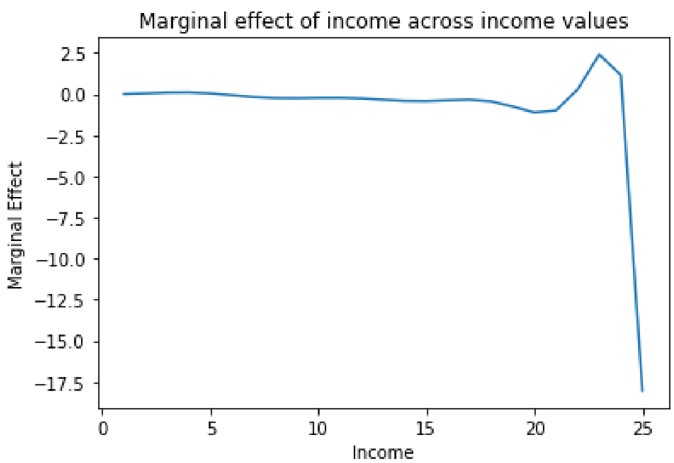
Out[526]:
```
array([ 8.33052123e-06,  3.86413357e-05,  2.27267669e-04,  7.25124016e-04,
        1.22065917e-03, -6.62426809e-04,  1.34765206e-04, -1.45754642e-05,
        9.20230578e-07, -3.41025761e-08,  6.88602186e-10, -5.85313884e-12])
```

In [527]:
```
print('The AME of income is {}'.format(ame))
```

```
The AME of income is -1.837827013750939
```

In [528]:
```
#graph marginal effect
def get_marginal_effect(x):
    y = 0
    for i in range(12):
        y += coef_degree_12[i] * (i+1) * (x ** (i))
    return y

marginal_effects = []
for x in range(1,27):
    marginal_effects.append(get_marginal_effect(x))
```

In [529]:
```
income_values = np.arange(1,26)
marginal_income = pd.DataFrame(zip(list(income_values), marginal_effects),
                               columns = ['income_value','marginal_effect'])
plt.plot(marginal_income['income_value'], marginal_income['marginal_effect'])
plt.title('Marginal effect of income across income values')
plt.xlabel('Income')
plt.ylabel('Marginal Effect');
```



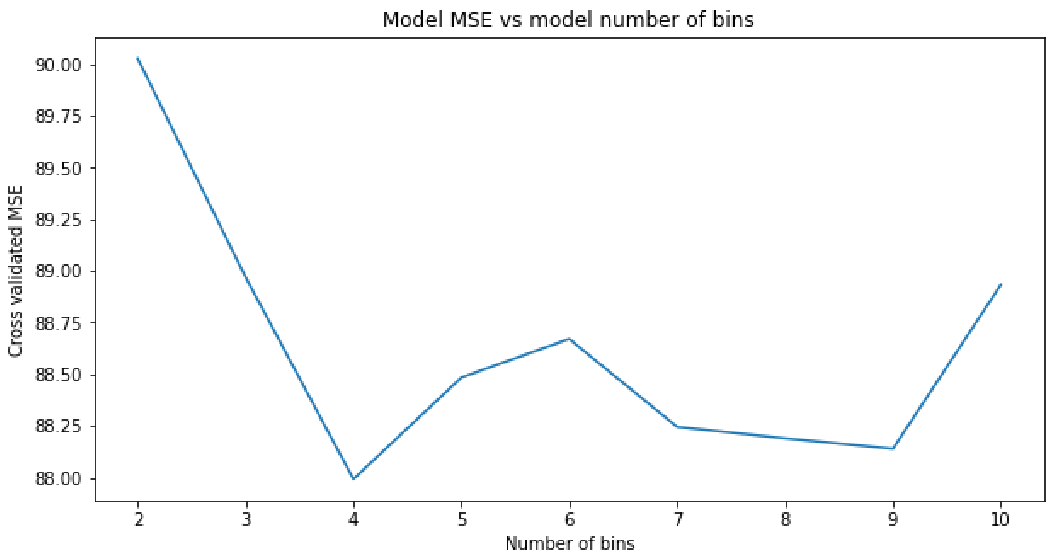
2)*(20 points) Fit a step function to predict egalit_scale as a function of income06, and perform 10-fold cross-validation to choose the optimal number of cuts. Plot the fit and interpret the results.*

In [530]:
```
def find_best_bin(bins = np.arange(2, 11)):

    cv = {}
    min_error = 10000000
    best_bin = 0

    for bin in bins:
        #cut data into bins
        step, bins = pd.cut(train.income06, bins=bin, retbins = True)
        #create dummy variables
        steps_dummies = pd.get_dummies(step)
        #combine y and x into one data frame
        step_df = pd.concat([train.egalit_scale, steps_dummies], axis = 1)
        x = np.array(step_df[step_df.columns[2:]])
        y = np.array(step_df['egalit_scale'])
        lr = LinearRegression()
        error = np.sum(-cross_val_score(lr, x , y, cv=10, scoring='neg_mean_squared_erro
r'))/10
        cv[bin] = error
        if error < min_error:
            min_error = error
            best_bin = bin

    return best_bin, cv
```

In [531]:
```
find_best_bin()
```

Out[531]:
```
(4,
 {2: 90.02781799497726,
  3: 88.97028485125078,
  4: 87.99268202181436,
  5: 88.48448329543504,
  6: 88.67079198404846,
  7: 88.24549422096784,
  8: 88.19032257110334,
  9: 88.13960580789443,
  10: 88.93320277814779})
```

In [532]:
```
#Plot the MSE graph
bins_list = sorted(find_best_bin()[1].items())
bins, mse = zip(*bins_list)
plt.figure(figsize=(10,5))
plt.plot(bins, mse)
plt.xlabel('Number of bins')
plt.ylabel('Cross validated MSE')
plt.title('Model MSE vs model number of bins');
```



The above graph shows that the optimal number of bins = 4. This means that when we break income into four different groups, the linear regression model will predict egalit_scale with the lowest error.

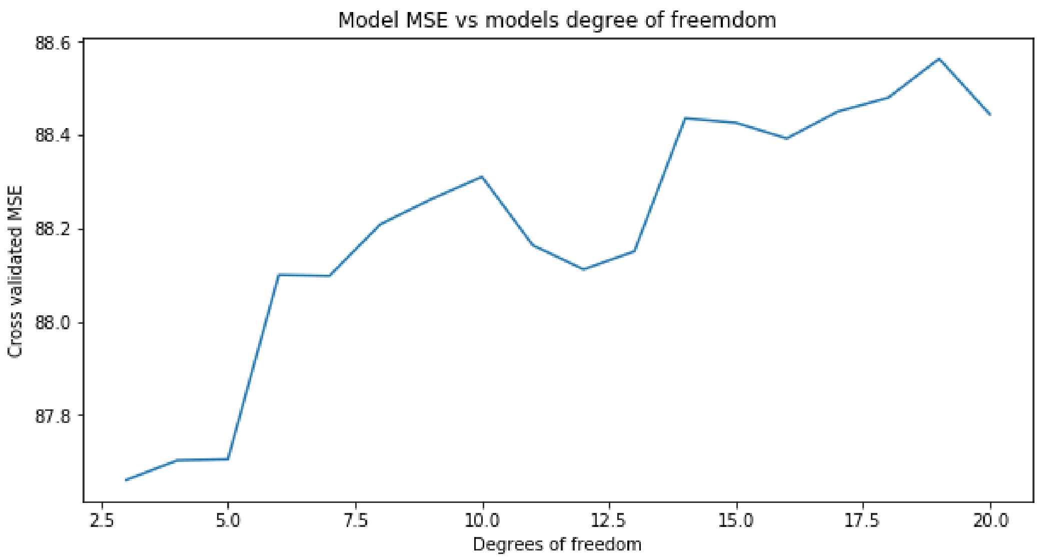
3)*(20 points) Fit a natural regression spline to predict egalit_scale as a function of income06. Use 10-fold cross-validation to select the optimal number of degrees of freedom, and present the results of the optimal model.*

In [533]:
```
def find_best_deg(X, y):
    cv = {}
    min_error = 10000000
    min_deg = 0
    degrees = np.arange(3, 21)
    x = 0

    for deg in degrees:
        #transform to cubic spline
        x = dmatrix("cr(X,df = {})".format(deg), {"X": X}, return_type='dataframe')
        # Fit regression
        lr = LinearRegression()
        #get error
        error = np.sum(-cross_val_score(lr, x, y, cv=10, scoring='neg_mean_squared_erro
r'))/10
        cv[deg] = error
        if error < min_error:
            min_error = error
            min_deg = deg

    return min_deg, cv
```

In [534]:
```
find_best_deg(X,y)
```

Out[534]:
```
(3,
 {3: 87.66117598417557,
  4: 87.7027438611445,
  5: 87.70523439574966,
  6: 88.10043282440009,
  7: 88.09833195179264,
  8: 88.20904399518508,
  9: 88.26253460814083,
  10: 88.31086968848001,
  11: 88.16429451853226,
  12: 88.11220442409994,
  13: 88.15088832132531,
  14: 88.43623469335596,
  15: 88.42653388451811,
  16: 88.39292563579053,
  17: 88.45039559291578,
  18: 88.48043046470737,
  19: 88.5637800942234,
  20: 88.44447802979225})
```

In [535]:
```
#Plot the MSE graph
spline_list = sorted(find_best_deg(X,y)[1].items())
degree, mse = zip(*spline_list)
plt.figure(figsize=(10,5))
plt.plot(degree, mse)
plt.xlabel('Degrees of freedom')
plt.ylabel('Cross validated MSE')
plt.title('Model MSE vs models degree of freemdom');
```



The optimal model is when degree of freedom = 3. The MSE of that model is 87.66117598417557.

Egalitarianism and everything

4)*(20 points total) Estimate the following models using all the available predictors (be sure to perform appropriate data pre-processing (e.g., feature standardization) and hyperparameter tuning (e.g. lambda for PCR/PLS, lambda and alpha for elastic net). Also use 10-fold cross-validation for each model to estimate the model's performance using MSE):*

In [536]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1481 entries, 0 to 1480
Data columns (total 45 columns):
age                   1481 non-null int64
attend                1481 non-null object
authoritarianism      1481 non-null int64
black                 1481 non-null object
born                  1481 non-null object
childs                1481 non-null int64
colath                1481 non-null object
colrac                1481 non-null object
colcom                1481 non-null object
colmil                1481 non-null object
colhomo               1481 non-null object
colmslm               1481 non-null object
con_govt              1481 non-null int64
degree                1481 non-null object
egalit_scale          1481 non-null int64
evangelical           1481 non-null object
grass                 1481 non-null object
happy                 1481 non-null object
hispanic_2            1481 non-null object
homosex               1481 non-null object
income06              1481 non-null int64
marital               1481 non-null object
mode                  1481 non-null object
news                  1481 non-null object
owngun                1481 non-null object
partyid_3             1481 non-null object
polviews              1481 non-null object
pornlaw2              1481 non-null object
pray                  1481 non-null object
pres08                1481 non-null object
reborn_r              1481 non-null object
relig                 1481 non-null object
science_quiz          1481 non-null int64
sex                   1481 non-null object
sibs                  1481 non-null int64
social_connect        1481 non-null int64
social_cons3          1481 non-null object
south                 1481 non-null object
spend3                1481 non-null object
teensex               1481 non-null object
tolerance             1481 non-null int64
tvhours               1481 non-null int64
vetyears              1481 non-null object
wordsum               1481 non-null int64
zodiac                1481 non-null object
dtypes: int64(12), object(33)
memory usage: 520.7+ KB
```

In [537]:
```
#Create dummies for categorical varaibles

def create_dummies (df, new_df):
    for predictor in df:
        if df[predictor].dtypes == object:
            temp = pd.get_dummies(df[predictor])
            temp = temp.drop(temp.columns[0], axis=1)
            new_df = pd.concat([new_df, temp], axis=1)
    return new_df
```

In [538]:
```
#Standardize data

scaler = MinMaxScaler(feature_range=(0,1))

def standardize (df):
    new_df = pd.DataFrame()
    for predictor in df:
        if df[predictor].dtypes == 'int64':
            column = df[predictor].values.reshape(-1,1)
            scaler.fit(column)
            new_df[predictor] = scaler.transform(column).reshape(1,-1)[0]
    return new_df
```

In [539]:
```
#transform and standardize X and Y
X_train = standardize(train)
y_train = X_train['egalit_scale']
X_train = create_dummies(train, X_train)
X_train.drop('egalit_scale', axis=1, inplace=True)
X_test = standardize(test)
y_test = X_test['egalit_scale']
X_test = create_dummies(test, X_test)
X_test.drop('egalit_scale', axis=1, inplace=True)
```

In [540]:
```
X_train.head(2)
```

Out[540]:

	age	authoritarianism	childs	con_govt	income06	science_quiz	sibs	social_connect	tolerance	tvhours
0	0.042254	0.571429	0.00	1.000000	1.000000	0.7	0.066667	0.416667	0.666667	0.125
1	0.338028	0.571429	0.25	0.333333	0.916667	1.0	0.033333	0.416667	0.866667	0.125

2 rows × 102 columns

a. *(5 points) Linear regression*

In [541]:
```
lr = LinearRegression()
np.sum(-cross_val_score(lr, X_train, y_train, cv=10, scoring='neg_mean_squared_error'))/
10
```

Out[541]: 0.05520732886554378

In [542]:
```
lr = lr.fit(X_train, y_train)
```

b. *(5 points) Elastic net regression*

In [543]:
```
#define a function to find the best lambda and alpha
def find_best_para():
    min_mse = 10000
    best_model = None
    best_lambda = None
    best_alpha = None

    for i in np.arange(0.1,1,0.1):
        for j in np.arange(0.1,1,0.1):
            elastic = ElasticNet(l1_ratio=i, alpha=j)
            mse = np.sum(-cross_val_score(elastic, X_train, y_train, cv=10,
                                          scoring='neg_mean_squared_error'))/10
            if mse < min_mse:
                min_mse = mse
                best_model = elastic
                best_lambda = i
                best_alpha = j

    return best_lambda, best_alpha, min_mse
```

In [544]:
```
a,b,c = find_best_para()
```

In [545]:
```
"The best model has lambda = {}, alplha ={}, \
and the MSE of this model is equal to {}".format(a,b,c)
```

Out[545]:
```
'The best model has lambda = 0.1, alplha =0.1, and the MSE of this model is equal to 0.
05980777651982657'
```

In [546]:
```
elastic = ElasticNet(l1_ratio=0.1, alpha=0.1).fit(X_train, y_train)
```

c. *(5 points) Principal component regression*

In [547]:
```
def find_component_pca():
    min_mse = 10000
    number = 0
    for i in np.arange(1,21,1):
        pca = PCA(n_components = i)
        pc = pca.fit_transform(X_train)
        mse = np.sum(-cross_val_score(lr, pc, y_train, cv=10,
                                      scoring='neg_mean_squared_error'))/10
        if mse < min_mse:
            min_mse = mse
            number = i
    return i, min_mse
```

In [548]:
```
find_component_pca()
```

Out[548]:
```
(20, 0.05637811069577927)
```

In [549]:
```
pca = PCA(n_components = 20)
pc = pca.fit_transform(X_train)
lr_pca = LinearRegression().fit(pc, y_train)
```

We see that the best model has components = 20 (choosing among 1-20) and MSE =0.056623.

d. *(5 points) Partial least squares regression*

In [550]:
```
def find_component_pls():
    min_mse = 10000
    number = 0
    for i in np.arange(1,21,1):
        pls = PLSRegression(n_components=i)
        mse = np.sum(-cross_val_score(pls, X_train, y_train, cv=10,
                                      scoring='neg_mean_squared_error'))/10
        if mse < min_mse:
            min_mse = mse
            number = i
    return i, min_mse
```

In [551]:
```
find_component_pls()
```

Out[551]:
```
(20, 0.05486714606062361)
```

We see that the best PLS model also has components = 20 (choosing among 1-20), its MSE =0.054867.

In [552]:
```
pls = PLSRegression(n_components=20)
pls.fit(X_train, y_train)
```

Out[552]:
```
PLSRegression(copy=True, max_iter=500, n_components=20, scale=True, tol=1e-06)
```

5)*(20 points) For each final tuned version of each model fit, evaluate feature importance by generating feature interaction plots. Upon visual presentation, be sure to discuss the substantive results for these models and in comparison to each other (e.g., talk about feature importance, conditional effects, how these are ranked differently across different models, etc.).*

In [624]:
```
X_train.head(1)
```

Out[624]:

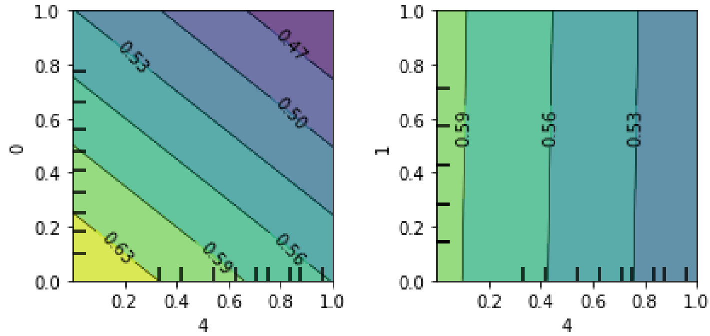
	age	authoritarianism	childs	con_govt	income06	science_quiz	sibs	social_connect	tolerance	tvhours
0	0.042254	0.571429	0.0	1.0	1.0	0.7	0.066667	0.416667	0.666667	0.125

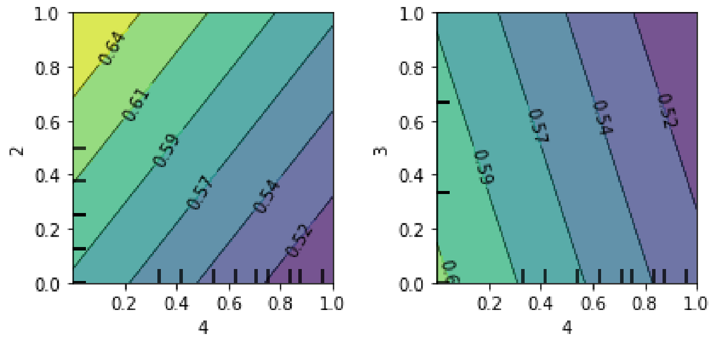
1 rows × 102 columns

In [590]:
```
features = []
for i in range(102):
    features.append((4,i))
```

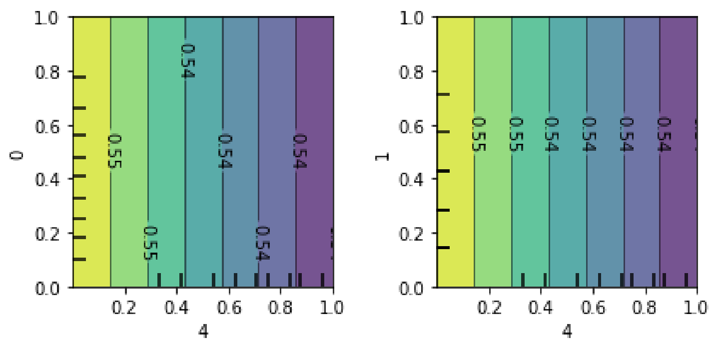
In [630]:
```
#There is no package for plotting feature interaction plot in python
#I will manually plot the interaction between
#income06 and 4 other variables for different models
```

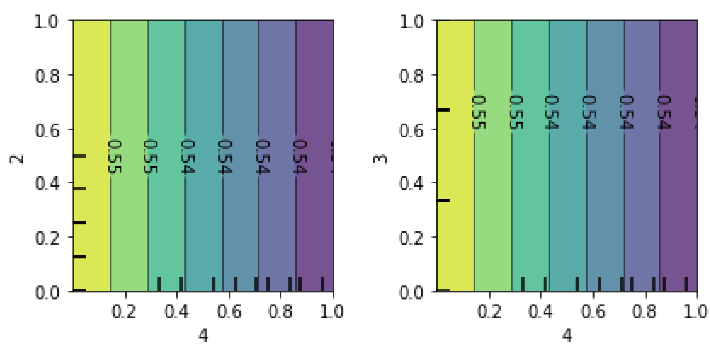
In [628]:
```
#linear regression
#0=age, 1=authoritarianism, 2= childs, 3=con_govt, 4 = income06
plot_partial_dependence(lr, X_train, [(4,0), (4,1)])
plot_partial_dependence(lr, X_train, [(4,2), (4,3)])
```



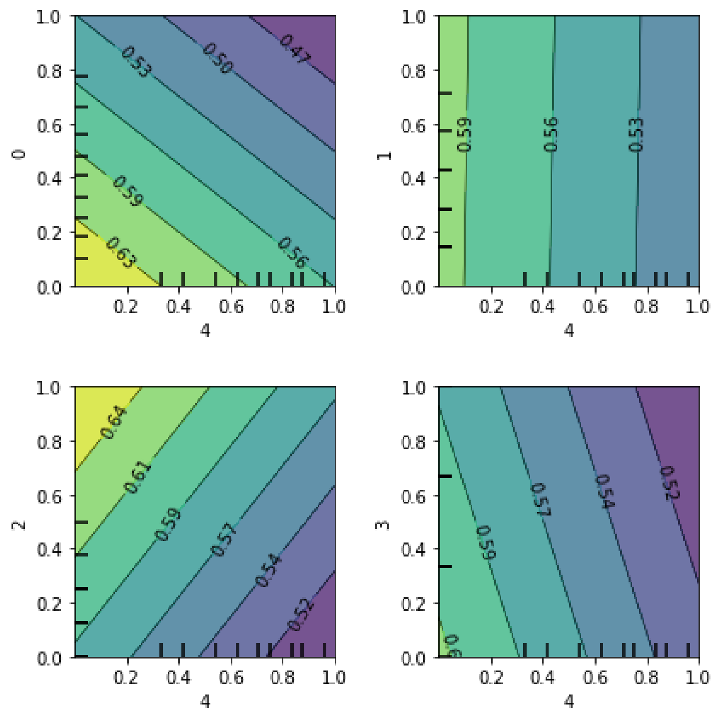


In [632]:
```
#elastic net
#0=age, 1=authoritarianism, 2= childs, 3=con_govt, 4 = income06
plot_partial_dependence(elastic, X_train, [(4,0), (4,1)])
plot_partial_dependence(elastic, X_train, [(4,2), (4,3)])
```





In [633]:
```
#pls
#0=age, 1=authoritarianism, 2= childs, 3=con_govt, 4 = income06
plot_partial_dependence(pls, X_train, [(4,0), (4,1)])
plot_partial_dependence(pls, X_train, [(4,2), (4,3)])
```



0=age, 1=authoritarianism, 2= childs, 3=con_govt, 4 = income06.
We see that for pls and linear regression, there seems to exist interactio nbetween income and age, income and childs, and income and congovt. No significant interaction is plotted for income and authoritarianism.
However, when using elastic net, we see that all interactions disappeared.