Homework 4

Chia-Yun Chang ¶

In [125]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from tabulate import tabulate
import math
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import Ridge
from sklearn.linear_model import LassoCV
from sklearn.linear_model import ElasticNetCV
from sklearn.metrics import mean_squared_error as mse
from sklearn.model_selection import GridSearchCV
from sklearn.base import BaseEstimator
import sklearn
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression
```

Question 1

In [13]:
```
train = pd.read_csv('gss_train.csv')
test = pd.read_csv('gss_test.csv')
x_train = train.income06
x_test = test.income06
y_train = train.egalit_scale
y_test = test.egalit_scale
```

In [14]:
```
class PolynomialRegression(BaseEstimator):
    def __init__(self, deg=None):
        self.deg = deg

    def fit(self, X, y, deg=None):
        self.model = LinearRegression(fit_intercept=False)
        self.model.fit(np.vander(X, N=self.deg + 1), y)

    def predict(self, x):
        return self.model.predict(np.vander(x, N=self.deg + 1))

    @property
    def coef_(self):
        return self.model.coef_
```

In [21]:
```
m = PolynomialRegression()
degrees = np.arange(1, 30)
cv_model = GridSearchCV(m,param_grid={'deg': degrees}, scoring = 'neg_mean_
cv_model.fit(x_train, y_train);
```

In [22]:
```
cv_model.best_params_, cv_model.best_estimator_.coef_
```

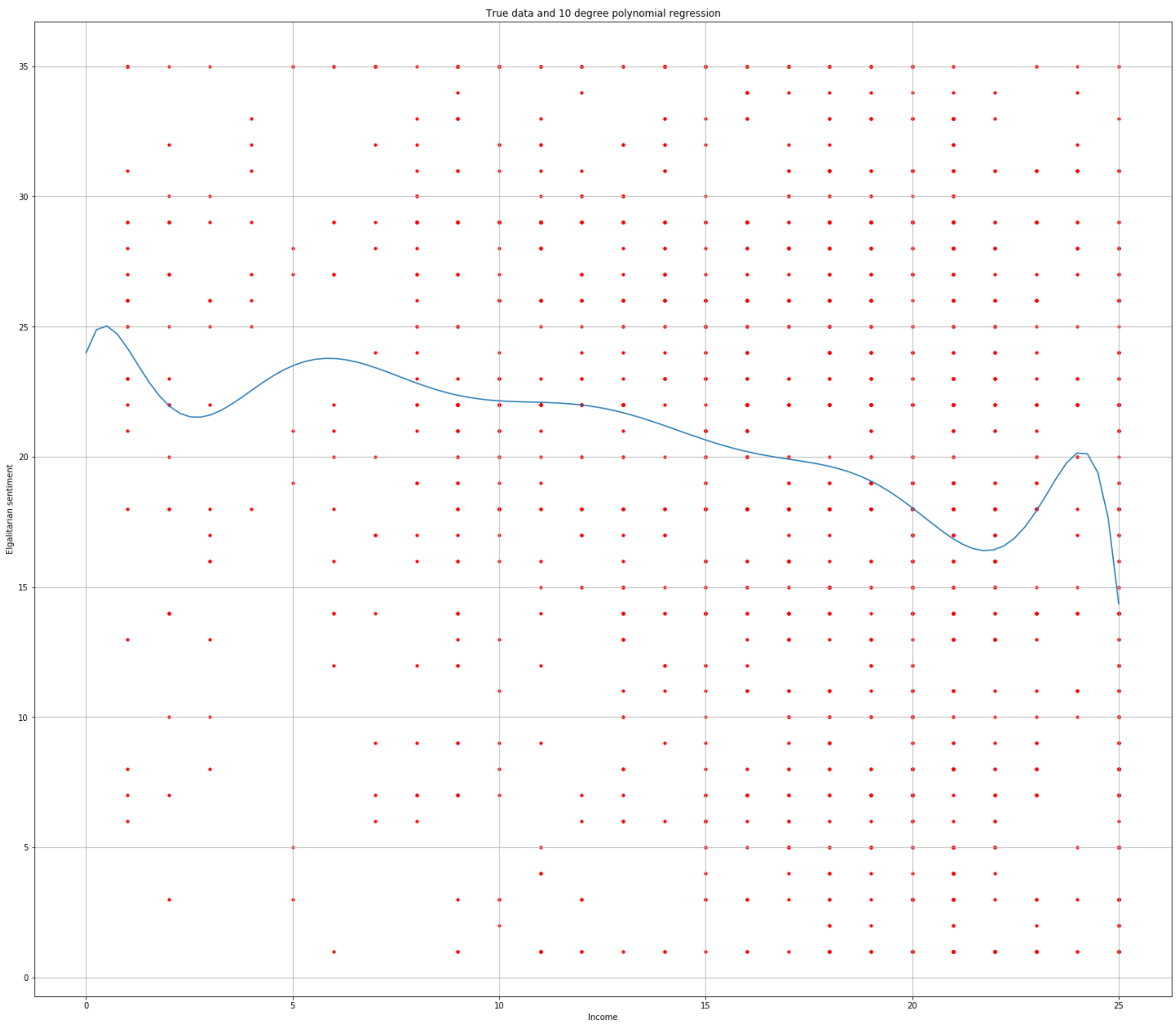
Out[22]:
```
({'deg': 10},
 array([-4.38896523e-09,  5.40568663e-07, -2.84426296e-05,  8.34108070e-0
4,
        -1.49053219e-02,  1.66442342e-01, -1.14208064e+00,  4.51519698e+0
0,
        -8.83699024e+00,  5.50620409e+00,  2.39852419e+01]))
```

In [55]:
```
X = np.linspace(0,25,100)
y = []
for x in X:
    '''
    yi = -4.38896523e-09 +  5.40568663e-07*x -2.84426296e-05*x**2 + 8.34108
        -1.49053219e-02*x**4 +  1.66442342e-01*x**5 -1.14208064e+00*x**6 +
        -8.83699024e+00*x**8 + 5.50620409e+00*x**9 +  2.39852419e+01*x**10

    '''
    yi = -4.38896523e-09*x**10 +  5.40568663e-07*x**9 -2.84426296e-05*x**8
        -1.49053219e-02*x**6 +  1.66442342e-01*x**5 -1.14208064e+00*x**4 +
        -8.83699024e+00*x**2 + 5.50620409e+00*x +  2.39852419e+01

    y.append(yi)
```

In [58]:
```
X = X.reshape(100,1)
```

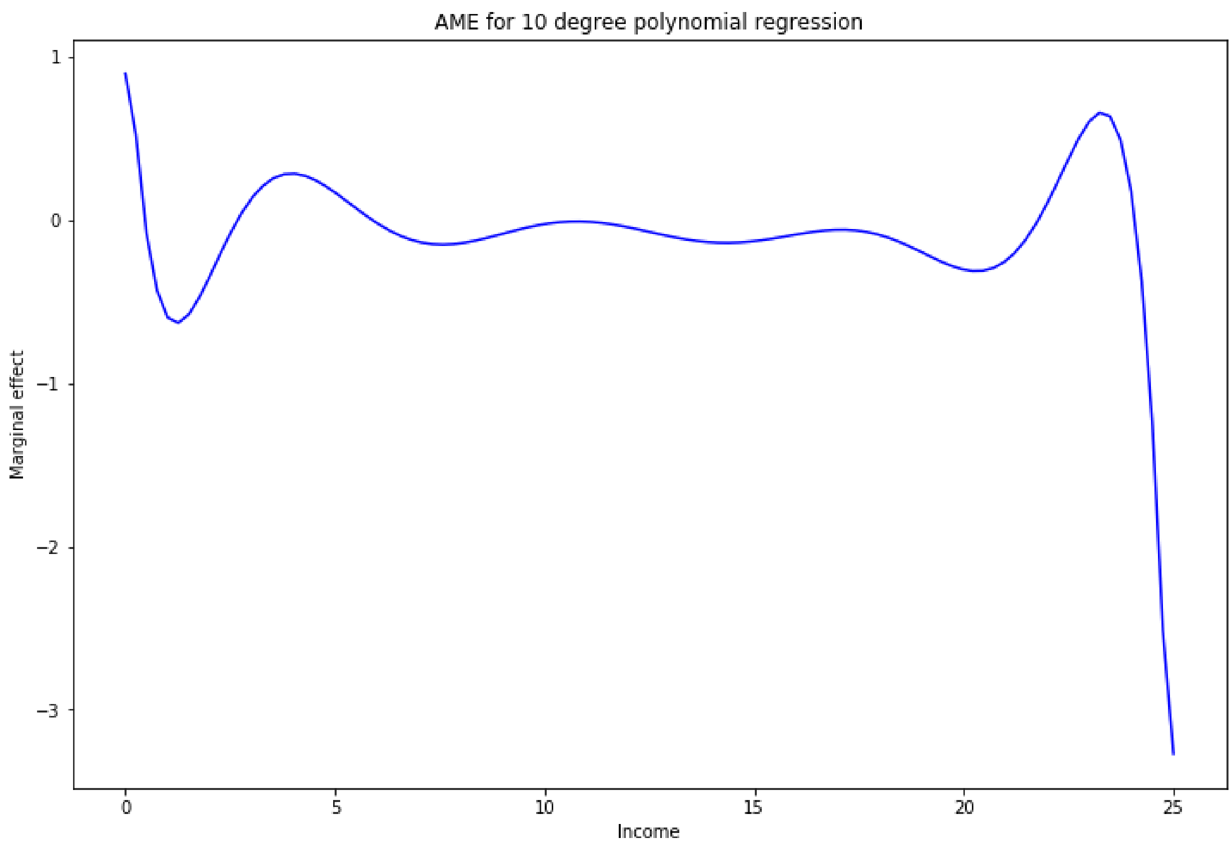
In [68]:
```
fig= plt.figure(figsize=(25,22))
ax1 = fig.add_subplot(111)

plt.plot(X,y)
ax1.scatter(x_train, y_train, s=10, c='r', marker="o", label='True Data')
plt.grid()

plt.title('True data and 10 degree polynomial regression')
plt.xlabel('Income')
plt.ylabel('Elgalitarian sentiment')
```

Out[68]: Text(0, 0.5, 'Elgalitarian sentiment')



In [71]:
```
ame_x = X
ame_y = np.gradient(y)
fig= plt.figure(figsize=(12,8))
plt.plot(ame_x, ame_y, c = 'blue')
plt.title('AME for 10 degree polynomial regression')
plt.xlabel('Income')
plt.ylabel('Marginal effect')
```

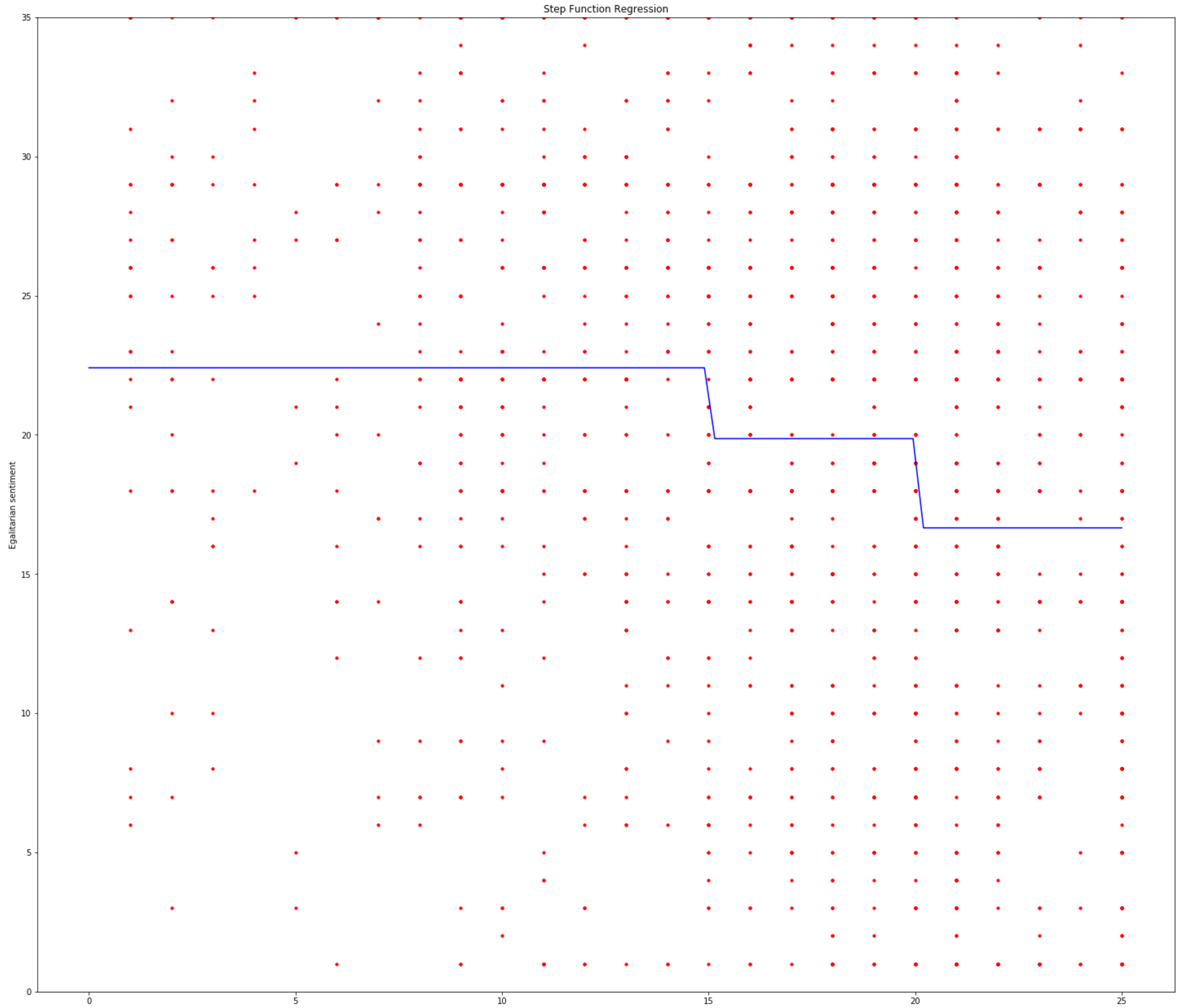
Out[71]:
```
Text(0, 0.5, 'Marginal effect')
```



The optimal fit is at degree 10. This is pretty high degree polynomial and is quite hard to interpret. The marginal effect averages at 0, which means there's not much shift in the output(egalitarian sentiment) when the income changes from one value to another(given that this is a discrete input).

Question 2

In [88]:
```
param = {'binner__n_bins': range(1, 11)}
m = GridSearchCV(Pipeline([('binner', KBinsDiscretizer()), ('linear',Linear
                        scoring='neg_mean_squared_error', refit=True, cv=10)
x_train = train.income06.values.reshape(-1, 1)
m.fit(x_train, y_train)
m2 = m.best_estimator_
y = m2.predict(X)

fig= plt.figure(figsize=(25,22))
ax1 = fig.add_subplot(111)
plt.plot(X, y, c='blue')
ax1.scatter(x_train, y_train, s=10, c='r', marker="o", label='True Data')
plt.ylim(0, 35)
plt.xlabel('Income')
plt.ylabel('Egalitarian sentiment')
plt.title('Step Function Regression')
plt.show()
```



As shown above, elalitarian sentiment is predicted to be sorted into 3 bins according to income. The difference between the bins are actually quite obvious; The step function performs slightly better than the 10 degree polynomial in terms of MSE, suggesting that the step regression could actually be a better model.

Question 4

In [103]:
```
train = pd.read_csv('gss_train.csv')
test = pd.read_csv('gss_test.csv')

maps = {}
for col in test.columns:
    d1, d2 = list(test[col]), list(train[col])
    encoder = LabelEncoder().fit(d1+d2)
    test[col] = encoder.transform(d1)
    train[col] = encoder.transform(d2)
    maps[col] = encoder.classes_

train, test = train.dropna(axis=0),test.dropna(axis=0)

x_train, y_train = train.drop('egalit_scale', axis=1), train['egalit_scale'
x_test, y_test = test.drop('egalit_scale', axis=1), test['egalit_scale']

x_train, y_train, x_test, y_test = [i.to_numpy() for i in [x_train, y_train
y_train, y_test = (i.reshape(-1, 1) for i in (y_train, y_test))
```

In [106]:
```
x1, x2 = StandardScaler(), StandardScaler()
x1, x2 = x1.fit(x_train), x2.fit(x_test)
xtr, xte = x1.transform(x_train), x2.transform(x_test)
```

In [110]:
```
# Linear Regression
lrcv = GridSearchCV(LinearRegression(), {}, scoring='neg_mean_squared_error

lrcv.fit(xtr, y_train)

best_lr = lrcv.best_estimator_
lr_err = mse(y_test, best_lr.predict(xte))
print("Best Linear regression model test MSE:", lr_err)
```

```
Best Linear regression model test MSE: 63.92805708826053
```

In [131]:
```
#Elastic Net
l1 = np.arange(0.1, 1, 0.1).tolist()
elcv = ElasticNetCV(l1_ratio=l1, n_alphas=10, cv=10)
y_train = y_train.reshape(-1,)
elcv.fit(xtr, y_train)
el_err = mse(y_test, elcv.predict(xte))
print("Best ElasticNet model test MSE: ", el_err)
print("lambda = :", elcv.alpha_, "  alpha =", elcv.l1_ratio_)
```

```
Best ElasticNet model test MSE:  62.61671271747407
lambda = : 0.1646097187236135   alpha = 0.6
```

In [132]:

```
#PCR
pcr = Pipeline([('pca', PCA()), ('ridge', Ridge())])
param_grid = {'pca__n_components':np.arange(2, 24, 2), 'ridge__alpha':[0.01
pcacv = GridSearchCV(pcr, param_grid, scoring='neg_mean_squared_error', cv=
pcacv.fit(xtr, y_train)
best_pca = pcacv.best_estimator_
n = pcacv.best_params_['pca__n_components']
lambda_ = pcacv.best_params_['ridge__alpha']
pca_err = mse(y_test, best_pca.predict(xte))
print("Best PCR model test MSE for best model:", pca_err)
print("n = :", n, "  lambda =",lambda_ )
```

```
Best PCR model test MSE for best model: 62.907478007216454
n = : 20   lambda = 0.1
```

In [133]:

```
#PLS
pls = PLSRegression()
plscv = GridSearchCV(pls, param_grid={'n_components':np.arange(2, 21, 2)},
plscv.fit(xtr, y_train)
best_pls = plscv.best_estimator_
n = plscv.best_params_['n_components']
pls_err = mse(y_test, best_pls.predict(xte))

print("best PLS model test MSE :", pls_err)
print("n = :", n)
```

```
best PLS model test MSE : 63.92770935062381
n = : 12
```