Homework 6

Yue Peng

Use the data from the iod.csv file you analyzed in the previous homework to create a model for GFR that considers nonlinear transformations (polynomials, step functions, splines, generalized additive models) for continuous predictors. Try each type of nonlinear function and give statistical support (e.g, by using a hypothesis test) to your final choice. If the function is nonparametric or hard to interpret, try to represent it using a function that can be interpreted straightforwardly.

We will now perform lasso regression to predict GFR with feature selection on the data. We now perform *10-fold* cross-validation to choose the best alpha, refit the model, and compute the associated test error

We use the *LassoCV* object that sets its λ parameter automatically from the data by internal cross-validation (i.e. it performs cross-validation on the training data it receives).

```
The test data MSE is 149.89
The test data R-squared is 0.72
```

$$GFR = \$ - 13.64599 \times cys + 5.95952 \times BLACK + -4.30164 \times SCR +$$
$$-3.46714 \times FEMALE + 2.8663 \times ALB + 1.59638 \times PHOS + -0.20604 \times SUN +$$
$$0.12258 \times HB + 0.0848 \times UUN + -0.08286 \times AGE + 0.05791 \times SODIUM +$$
$$-0.01942 \times HEIGHT + 0.01148 \times GLUC + 0.00694 \times BMI + 0.0034 \times UCRE +$$
$$-0.00271 \times TCHOL + -0.00032 \times upro + 0.0 \times WEIGHT + 0.0 \times MAP +$$
$$-0.0 \times BSA + 0.0 \times DBP + 0.0 \times SBP + 0.0 \times Diabetes$$

Also, the lasso has a substantial advantage over ridge regression in that the resulting coefficient estimates are sparse. Here we see that the coefficient estimates
of **WEIGHT**, **MAP**, **BSA**, **DBP**, **SBP** and **Diabetes** are exactly zero.

The other factors are predictive. This model predicts the data quite well with *MSE = 149.89*and *R-Squared = 0.72*

We selected some significant variables whose absolute value of their coefficient are greater than one.

	SCR	ALB	PHOS	BLACK	FEMALE	cys
2164	2.2	4.4	3.2	1	0	2.23
1181	2.6	4.5	3.0	1	0	1.33
2485	2.7	3.2	4.1	1	1	2.83
1004	1.2	4.1	2.9	0	1	1.23
1953	2.4	4.3	3.1	1	1	1.94

We could know that **cys**, **ALB**, **PHOS** and **SCR** are continuous predictors.

Quadratic polynomial

We have done quadratic transformation on all continuous variables.

```
The test data MSE is 118.37
The test data R-squared is 0.78
```

Cubic polynomial

We have done cubic transformation on all continuous variables

```
The test data MSE is 109.13
The test data R-squared is 0.79
```

Deciding on a degree

We can do this using the **anova_lm()** function, which performs an analysis of variance (ANOVA, using an F-test) in order to test the null hypothesis that a model Model 1 is sufficient to explain the data against the alternative hypothesis that a more complex model Model 2 is required.

	df_resid	ssr	df_diff	ss_diff	F	Pr(>F)
0	1088.0	184952.160654	0.0	NaN	NaN	NaN
1	1078.0	131444.445275	10.0	53507.715379	47.842629	3.438929e-79
2	1058.0	120039.870896	20.0	11404.574380	5.098562	2.671357e-12
3	1023.0	115066.228375	35.0	4973.642520	1.270589	1.363428e-01
4	967.0	108150.329846	56.0	6915.898529	1.104229	2.827319e-01

The p-value comparing the linear Model 1 to the quadratic Model 2 is essentially zero (10e−78), indicating that a linear fit is not sufficient. Similarly the p-value comparing the quadratic Model 2 to the cubic Model 3 is very low (<10e−11), so the quadratic fit is also insufficient. The p-value comparing the cubic and degree-4 polynomials, Model 3 and Model 4, is approximately 0.01 while the degree-5 polynomial Model 5 seems unnecessary because its p-value is 0.28. Hence, either a cubic or a quartic

polynomial appear to provide a reasonable fit to the data, but lower- or higher-order models are not justified.

Step function

First, we deal with the **cys** predictor

	coef	std err	z	P>|z|	[0.025	0.975]
const	2.858e+13	1.64e+14	0.174	0.862	-2.93e+14	3.51e+14
(0.515, 1.71]	-2.858e+13	1.64e+14	-0.174	0.862	-3.51e+14	2.93e+14
(1.71, 2.9]	-2.858e+13	1.64e+14	-0.174	0.862	-3.51e+14	2.93e+14
(2.9, 4.09]	-2.858e+13	1.64e+14	-0.174	0.862	-3.51e+14	2.93e+14
(4.09, 5.28]	-2.858e+13	1.64e+14	-0.174	0.862	-3.51e+14	2.93e+14

Then, we deal with the **ALB** predictor

	coef	std err	z	P>|z|	[0.025	0.975]
const	35.8918	0.811	44.241	0.000	34.302	37.482
(2.798, 3.425]	-5.0783	2.567	-1.978	0.048	-10.110	-0.047
(3.425, 4.05]	2.7367	1.229	2.226	0.026	0.327	5.146
(4.05, 4.675]	15.5430	1.047	14.850	0.000	13.492	17.594
(4.675, 5.3]	22.6905	1.819	12.476	0.000	19.126	26.255

Then, we deal with the **PHOS** predictor

	coef	std err	z	P>|z|	[0.025	0.975]
const	30.3083	5.971	5.076	0.000	18.606	42.011
(1.289, 4.125]	21.1079	5.996	3.520	0.000	9.356	32.860
(4.125, 6.95]	0.7430	6.077	0.122	0.903	-11.167	12.653
(6.95, 9.775]	-21.7953	17.385	-1.254	0.210	-55.870	12.279
(9.775, 12.6]	30.2527	17.385	1.740	0.082	-3.822	64.327

Then, we deal with the **SCR** predictor

	coef	std err	z	P>|z|	[0.025	0.975]
const	8.471e+13	1.87e+14	0.454	0.650	-2.81e+14	4.5e+14
(0.794, 2.35]	-8.471e+13	1.87e+14	-0.454	0.650	-4.5e+14	2.81e+14
(2.35, 3.9]	-8.471e+13	1.87e+14	-0.454	0.650	-4.5e+14	2.81e+14
(3.9, 5.45]	-8.471e+13	1.87e+14	-0.454	0.650	-4.5e+14	2.81e+14
(5.45, 7.0]	-8.471e+13	1.87e+14	-0.454	0.650	-4.5e+14	2.81e+14

According to the tables above, we choose **ALB** variables to analyze since its p-values are all less than 0.05.

```
The test data MSE is 466.83
```

Since the MSE is quite large, it is not flexible and efficient to use step function here.

Spline

First, we used 3 knots and 3 degrees (cubic spline). The chosen variable is still **ALB** according to the analysis above.

```
The test data MSE is 468.96
```

Then we used 3 knots and 6 degrees. The chosen variable is still **ALB** according to the analysis above.

```
The test data MSE is 467.62
```

Then we used 5 knots and 3 degrees. The chosen variable is still **ALB** according to the analysis above.

```
The test data MSE is 466.59
```

The **MSE** is quite large compared to polynomial and decreases not obviously with higher degree or more knots.

We decided to move to next methods.

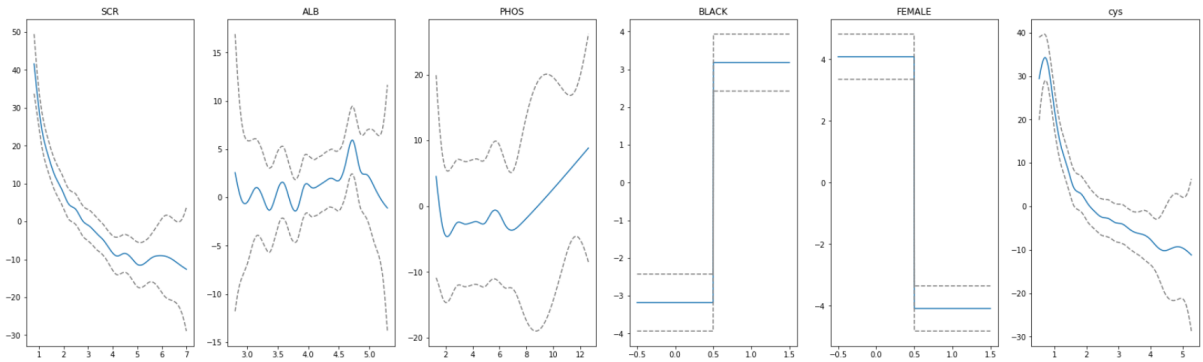
GAM

Model Statistics	
edof	54.338
AIC	8141.254
AICc	8147.268
GCV	104.837
loglikelihood	-4015.288
deviance	1038.662
scale	95.499

Pseudo-R^2	
explained_deviance	0.821
McFadden	0.373
McFadden_adj	0.365

We fit a GAM with 25 splines to use in each of the smooth function that is going to be fitted. The penalization term is 0.6 that is multiplied to the second derivative in the overall objective function. And no constraints.

The model is quite good with lowest AIC and MSE (92.71) and highest R-squared (0.821)



It is clear that some features like **SCR** and **cys** have a fairly simple linear relationship with the **GFR** variable. There are about four features including two non-continuous variables that seem to have strong non-linear relationships though.

Also, it has the smallest MSE and AIC among four methods we had tested.

In all, the GAM method would be used in further analysis.

Appendix

```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
# from IPython.display import Markdown, display

from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso, LassoCV
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold, cross_val_score
from sklearn.preprocessing import scale
def pretty_print_linear(coefs, names = None, sort = False):
    if names == None:
        names = ["X%s" % x for x in range(len(coefs))]
    lst = zip(coefs, names)
    if sort:
        lst = sorted(lst,  key = lambda x:-np.abs(x[0]))
    return " + ".join("%s \times %s" % (round(coef, 5), name)
                                for coef, name in lst)
```

```
# def printmd(string):
#     display(Markdown(string))
data = pd.read_csv('iodatadev.csv')
vars = ["WEIGHT", "BMI", "GFR", "UCRE", "UUN", "UPHO", "SUN", "SCR", "TCHOL
", "ALB", "HBA1C", "PHOS",
        "TRIG", "LDL", "HDL", "HB", "MAP", "upro", "BSA", "SODIUM", "GLUC",
"BLACK", "HEIGHT", "AGE",
        "FEMALE", "cys", "DBP", "SBP", "CRP", "Diabetes", "hbpstatus"]
data = data[vars]
data.shape
for i in vars:
    print('%s has %d missing values' % (i, sum(data[i].isna())))
dat = data[["WEIGHT", "BMI", "GFR", "UCRE", "UUN", "SUN", "SCR", "TCHOL", "
ALB", "PHOS",
         "HB", "MAP", "upro", "BSA", "SODIUM", "GLUC", "BLACK", "HEIGHT", "A
GE",
        "FEMALE", "cys", "DBP", "SBP", "Diabetes"]]
dat = dat.dropna()
X = dat[["WEIGHT", "BMI", "UCRE", "UUN", "SUN", "SCR", "TCHOL", "ALB", "PHO
S",
         "HB", "MAP", "upro", "BSA", "SODIUM", "GLUC", "BLACK", "HEIGHT", "A
GE",
        "FEMALE", "cys", "DBP", "SBP", "Diabetes"]]
y = dat["GFR"]
print("Finally we keep %.2f of data points with keeping %d/30 predicors" %
(X.shape[0]/data.shape[0], X.shape[1]))
X_train, X_test , y_train, y_test = train_test_split(X, y, test_size=0.3, r
andom_state=123)
lassocv = LassoCV(alphas=None, cv=10, max_iter=100000, normalize=True)
lassocv.fit(X_train, y_train)
lasso = Lasso(max_iter=10000, normalize=True)
lasso.set_params(alpha=lassocv.alpha_)
lasso.fit(X_train, y_train)
print("The test data MSE is %.2f" % (mean_squared_error(y_test, lasso.predi
ct(X_test))))
print("The test data R-squared is %.2f" % (lasso.score(X_test, y_test)))
lasso.fit(X, y)
pretty_print_linear(lasso.coef_, list(X.columns), sort = True)
vars_selected = ["SCR", "ALB", "PHOS", "BLACK", "FEMALE", "cys"]
X_train = X_train[vars_selected]
```

```
X_test = X_test[vars_selected]
X_train.head()
from sklearn import linear_model
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree = 2)
X_train_poly = poly_reg.fit_transform(X_train[["cys", "ALB", "PHOS", "SCR
"]])
X_test_poly = poly_reg.fit_transform(X_test[["cys", "ALB", "PHOS", "SCR"]])
lin_reg_2 = linear_model.LinearRegression()
lin_reg_2.fit(X_train_poly, y_train)
print("The test data MSE is %.2f" % (mean_squared_error(y_test, lin_reg_2.p
redict(X_test_poly))))
print("The test data R-squared is %.2f" % (lin_reg_2.score(X_test_poly, y_t
est)))
poly_reg3 = PolynomialFeatures(degree = 3)
X_train_poly3 = poly_reg3.fit_transform(X_train[["cys", "ALB", "PHOS", "SCR
"]])
X_test_poly3 = poly_reg3.fit_transform(X_test[["cys", "ALB", "PHOS", "SCR
"]])
lin_reg_3 = linear_model.LinearRegression()
lin_reg_3.fit(X_train_poly3, y_train)
print("The test data MSE is %.2f" % (mean_squared_error(y_test, lin_reg_3.p
redict(X_test_poly3))))
print("The test data R-squared is %.2f" % (lin_reg_3.score(X_test_poly3, y_
test)))
import statsmodels.api as sm
X1 = PolynomialFeatures(1).fit_transform(X_train[["cys", "ALB", "PHOS", "SC
R"]])
X2 = PolynomialFeatures(2).fit_transform(X_train[["cys", "ALB", "PHOS", "SC
R"]])
X3 = PolynomialFeatures(3).fit_transform(X_train[["cys", "ALB", "PHOS", "SC
R"]])
X4 = PolynomialFeatures(4).fit_transform(X_train[["cys", "ALB", "PHOS", "SC
R"]])
X5 = PolynomialFeatures(5).fit_transform(X_train[["cys", "ALB", "PHOS", "SC
R"]])
fit_1 = fit = sm.GLS(y_train, X1).fit()
fit_2 = fit = sm.GLS(y_train, X2).fit()
fit_3 = fit = sm.GLS(y_train, X3).fit()
fit_4 = fit = sm.GLS(y_train, X4).fit()
fit_5 = fit = sm.GLS(y_train, X5).fit()
```

```
print(sm.stats.anova_lm(fit_1, fit_2, fit_3, fit_4, fit_5, typ=1))
df_cut, bins = pd.cut(X_train.cys, 4, retbins = True, right = True)
df_cut.value_counts(sort = False)
df_steps = pd.concat([X_train.cys, df_cut, y_train], keys = ['cys','cys_cut
s','GFR'], axis = 1)

# Create dummy variables for the cys groups
df_steps_dummies = pd.get_dummies(df_steps['cys_cuts'])

# Statsmodels requires explicit adding of a constant (intercept)
df_steps_dummies = sm.add_constant(df_steps_dummies)

fit1 = sm.GLM(df_steps.GFR, df_steps_dummies).fit()
fit1.summary().tables[1]
df_cut, bins = pd.cut(X_train.ALB, 4, retbins = True, right = True)
df_cut.value_counts(sort = False)
df_steps = pd.concat([X_train.ALB, df_cut, y_train], keys = ['ALB','ALB_cut
s','GFR'], axis = 1)

# Create dummy variables for the cys groups
df_steps_dummies = pd.get_dummies(df_steps['ALB_cuts'])

# Statsmodels requires explicit adding of a constant (intercept)
df_steps_dummies = sm.add_constant(df_steps_dummies)

fit2 = sm.GLM(df_steps.GFR, df_steps_dummies).fit()
fit2.summary().tables[1]
df_cut, bins = pd.cut(X_train.PHOS, 4, retbins = True, right = True)
df_cut.value_counts(sort = False)
df_steps = pd.concat([X_train.PHOS, df_cut, y_train], keys = ['PHOS','PHOS_
cuts','GFR'], axis = 1)

# Create dummy variables for the cys groups
df_steps_dummies = pd.get_dummies(df_steps['PHOS_cuts'])

# Statsmodels requires explicit adding of a constant (intercept)
df_steps_dummies = sm.add_constant(df_steps_dummies)

fit3 = sm.GLM(df_steps.GFR, df_steps_dummies).fit()
fit3.summary().tables[1]
```

```
df_cut, bins = pd.cut(X_train.SCR, 4, retbins = True, right = True)
df_cut.value_counts(sort = False)
df_steps = pd.concat([X_train.SCR, df_cut, y_train], keys = ['SCR','SCR_cut
s','GFR'], axis = 1)

# Create dummy variables for the cys groups
df_steps_dummies = pd.get_dummies(df_steps['SCR_cuts'])

# Statsmodels requires explicit adding of a constant (intercept)
df_steps_dummies = sm.add_constant(df_steps_dummies)

fit4 = sm.GLM(df_steps.GFR, df_steps_dummies).fit()
fit4.summary().tables[1]
df_cut, bins = pd.cut(X_test.ALB, 4, retbins = True, right = True)
df_cut.value_counts(sort = False)
df_steps = pd.concat([X_test.ALB, df_cut, y_test], keys = ['ALB','ALB_cuts
','GFR'], axis = 1)

# Create dummy variables for the cys groups
df_steps_dummies = pd.get_dummies(df_steps['ALB_cuts'])

# Statsmodels requires explicit adding of a constant (intercept)
df_steps_dummies = sm.add_constant(df_steps_dummies)
mean_squared_error(y_test, fit2.predict(df_steps_dummies))
from patsy import dmatrix

# Specifying 3 knots
transformed_x_train = dmatrix("bs(X_train.ALB, knots=(3, 4, 5), degree=3, i
nclude_intercept=False)",
                    {"X_train.ALB": X_train.ALB}, return_type='dataframe')
transformed_x_test = dmatrix("bs(X_test.ALB, knots=(3, 4, 5), degree=3, inc
lude_intercept=False)",
                    {"X_test.ALB": X_test.ALB}, return_type='dataframe')
# Build a regular linear model from the splines
fits1 = sm.GLM(y_train, transformed_x_train).fit()
fits1.params
mean_squared_error(y_test, fits1.predict(transformed_x_test))
# Specifying 6 degrees of freedom
transformed_x_train1 = dmatrix("bs(X_train.ALB, df=6, include_intercept=Fa
lse)",
                    {"X_train.ALB": X_train.ALB}, return_type='dataframe')
```

```
transformed_x_test2 = dmatrix("bs(X_test.ALB, df=6, include_intercept=Fals
e)",
                     {"X_test.ALB": X_test.ALB}, return_type='dataframe')
fits2 = sm.GLM(y_train, transformed_x_train1).fit()
fits2.params
mean_squared_error(y_test, fits2.predict(transformed_x_test2))
from pygam import LinearGAM
from pygam.utils import generate_X_grid
gam = LinearGAM().fit(X_train, y_train)
gam.summary()
mean_squared_error(y_test, gam.predict(X_test))
XX = generate_X_grid(gam)
plt.rcParams['figure.figsize'] = (28, 8)
fig, axs = plt.subplots(1, 6)
titles = X_train.columns
for i, ax in enumerate(axs):
    pdep, confi = gam.partial_dependence(XX, feature=i+1, width=.95)
    ax.plot(XX[:, i], pdep)
    ax.plot(XX[:, i], confi[0][:, 0], c='grey', ls='--')
    ax.plot(XX[:, i], confi[0][:, 1], c='grey', ls='--')
    ax.set_title(titles[i])

plt.show()
```