Coding Appendix to PS5 Q1, BUSN 41903

Professor Christian Hansen, TA Jianfei Cao

Submitted by Ginha Kim, Junho Choi

In [160]:
```
import pandas as pd
import numpy as np
import statsmodels.api as sts
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from numpy.random import randint, seed
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, KFold, RandomizedSearchCV
from sklearn.model_selection import LeaveOneOut
from sklearn.metrics import classification_report
```

In [377]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [29]:
```
myfilename = "D:/All/Documents/UChicago/2018-19/BoothMetrics/PS5/PS5Data/MROZ.csv"
df = pd.read_csv(myfilename, header=None, na_values='.')
df.columns = ['inlf', 'hours', 'kidslt6', 'kidsge6', 'age',
              'educ', 'wage', 'repwage', 'hushrs', 'husage',
              'huseduc', 'huswage', 'faminc', 'mtr', 'motheduc',
              'fatheduc', 'unem', 'city', 'exper', 'nwifeinc',
              'lwage', 'expersq']
```

In [30]:
```
seed = 60637
yname = ['inlf']
Xnames = ['kidslt6', 'kidsge6', 'age', 'educ', 'repwage', 'faminc', 'exper']

y = np.ravel(np.array(df[yname]))
X = df.loc[:, Xnames].values

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=int(253), random_state=60637)

data_split = (X_train, X_test, y_train, y_test)
```

In [490]:
```
loocv = LeaveOneOut()
```

1-(a). Kernel Estimator

Let's see the unique values for coarsely discrete variables. Initial glance suggests that it would be nearly impossible for us to use a very fine bandwidth becase product of unique values amounts to 580320, which is much, much more than the number of observations in the training set.

```
In [234]: print("Unique values of kidslt6:", np.unique(df['kidslt6']))
          print("Unique values of kidsge6:", np.unique(df['kidsge6']))
          print("Unique values of age:", np.unique(df['age']))
          print("Unique values of educ:", np.unique(df['educ']))
          print("Unique values of exper:", np.unique(df['exper']))
          hey = np.unique(df['kidslt6']).shape[0] * np.unique(df['kidsge6']).shape[0] * \
                np.unique(df['educ']).shape[0] * np.unique(df['age']).shape[0] * \
                np.unique(df['exper']).shape[0]
          print()
          print(hey)
```

```
Unique values of kidslt6: [0 1 2 3]
Unique values of kidsge6: [0 1 2 3 4 5 6 7 8]
Unique values of age: [30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
 54 55 56 57 58 59 60]
Unique values of educ: [ 5  6  7  8  9 10 11 12 13 14 15 16 17]
Unique values of exper: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 45]

580320
```

Define the function for cutting up the spaces

```
In [346]: X_df = pd.DataFrame(X_train)
          X_df.columns = Xnames
          y_df = pd.DataFrame(y_train)
```

```
In [311]: def cutup_avg(cutup_lst, X_df):
              lst_cnds = []
              h_lst = []
              for i, cut in enumerate(cutup_lst):
                  tgt = X_df.iloc[:, i]
                  tgt_min, tgt_max = tgt.min(), tgt.max()
                  length = tgt_max - tgt_min
                  h = length / cut
                  lb = tgt_min
                  ub = lb + h

                  for_one_val = []
                  for j in range(0, cut):
                      ub_cnd = (tgt <= ub)
                      if j == 0:
                          lb_cnd = (tgt >= lb)
                      else:
                          lb_cnd = (tgt > lb)
                      cnd = df.iloc[np.array(ub_cnd & lb_cnd), i].index

                      for_one_val.append(cnd)
                      ub += h
                      lb += h

                  lst_cnds.append(for_one_val)
                  h_lst.append(h)

              return lst_cnds, h_lst
```

Define the function for all possible combos of coordinates

```
In [326]: def coordinates_maker(lst):
              appender = []

              for x in itertools.product(range(1, max(lst)+1), repeat = 7):
                  appendthis = True
                  for i, kay in enumerate(x):
                      if lst[i] == max(lst):
                          continue
                      else:
                          if kay > lst[i]:
                              appendthis = False
                              break
                  if appendthis:
                      appender.append(x)

              return appender
```

Define the function for Gaussian kernel estimation; multivariate kernel will be just the product kernel

In [388]:

```
def gauss_ke(cutup_lst, X_df, y_df):
    lst_cnds, h_lst = cutup_avg(cutup_lst, X_df)
    coords = coordinates_maker(cutup_lst)

    cells_predy = []
    sse = 0
    for i, coord in enumerate(coords):
        gtg = True
        for j, entry in enumerate(coord):
            index = lst_cnds[j][entry-1]
            if j == 0:
                cell_index = index
            else:
                cell_index = cell_index.intersection(index)
                if len(cell_index) == 0:
                    gtg = False
                    break

        if not gtg:
            cells_predy.append("No obs")
            continue
        else:
            xcell = X_df.loc[cell_index, :].copy()
            ycell = y_df.loc[cell_index, 0].copy()
            xcell_demeaned = xcell - xcell.mean()
            for i in range(0, xcell.shape[1]):
                h = h_lst[i]
                which = xcell_demeaned.iloc[:, i]
                ## implementing Gaussian kernel
                kern = np.exp(-(which**2)/(2*h)) / np.sqrt(2 * np.pi * h)
                if i == 0:
                    product_kern = kern.copy()
                else:
                    product_kern = product_kern * kern

            numer = (ycell * product_kern).sum()
            denom = product_kern.sum()
            cell_predy = numer / denom
            sse_partial = ((ycell - cell_predy) ** 2).sum()
            sse += sse_partial
            cells_predy.append(cell_predy)

    return coords, cells_predy, sse
```

In [395]:

```
lst = [2, 2, 2, 2, 2, 2, 2]
a, b, c = gauss_ke(lst, X_df, y_df)
```

In [408]:

```
def sse_for_gaus_ke(cutup_lst, X_tr, y_tr, X_te, y_te):
    coords, cells_predy, sse = gauss_ke(cutup_lst, X_tr, y_tr)
    lst_cnds, h_lst = cutup_avg(cutup_lst, X_tr)

    total_sse = 0
    for row in range(X_te.shape[0]):
        row_case = []
        for col in range(X_df.shape[1]):
            mn = X_tr.iloc[:, col].min()
            mx = X_tr.iloc[:, col].max()
            if h_lst[col] == (mx - mn):
                row_case.append(1)
            else:
                case = X_te.iloc[row, col]
                if case >= mn and case <= mn + h_lst[col]:
                    row_case.append(1)
                else:
                    row_case.append(2)

        for i, coord in enumerate(coords):
            if coord == tuple(row_case):
                break

        found = cells_predy[i]
        if found == "No obs":
            total_sse += 0.5
        else:
            acty = y_te.iloc[row, 0]
            total_sse += (found - acty)**2

    return total_sse
```

In [407]:

```
gridcases = [x for x in itertools.product(range(1, 3), repeat=7)]
```

In [483]:

```
## takes a very long time!
best_h = 0
best_sse = np.inf
for cutup in gridcases:
    hvec = cutup_avg(cutup, X_df)[1]
    contender_sse = 0
    for train_index, test_index in loocv.split(X_df):
        X_, y_ = X_df.iloc[train_index, :], y_df.iloc[train_index, :]
        X_.index = range(0, X_.shape[0])
        y_.index = range(0, y_.shape[0])
        X__, y__ = X_df.iloc[test_index, :], y_df.iloc[test_index, :]
        sse_frac = sse_for_gaus_ke(cutup, X_, y_, X__, y__)
        contender_sse += sse_frac

    if contender_sse < best_sse:
        best_sse = contender_sse
        best_h = hvec
```

In [484]:

```
prev_best = 111.97723191660212
prev_best_h = [3.0, 4.0, 30.0, 12.0, 4.99, 94500.0, 45.0]
```

In [489]:

```
Xnames
```

Out[489]:

```
['kidslt6', 'kidsge6', 'age', 'educ', 'repwage', 'faminc', 'exper']
```

In [492]:

```
np.unique(X_df.repwage)
```

Out[492]:

```
array([0.  , 1.  , 1.13, 1.36, 1.44, 1.5 , 1.65, 1.8 , 1.85, 1.9 , 2.  ,
       2.1 , 2.2 , 2.22, 2.25, 2.26, 2.3 , 2.37, 2.4 , 2.5 , 2.57, 2.6 ,
       2.64, 2.7 , 2.74, 2.75, 2.76, 2.8 , 2.9 , 2.93, 2.95, 3.  , 3.05,
       3.08, 3.14, 3.21, 3.23, 3.25, 3.26, 3.27, 3.29, 3.3 , 3.35, 3.38,
       3.4 , 3.45, 3.5 , 3.58, 3.6 , 3.69, 3.75, 3.8 , 3.85, 3.87, 3.9 ,
       3.94, 3.95, 3.97, 4.  , 4.05, 4.07, 4.15, 4.19, 4.2 , 4.26, 4.3 ,
       4.32, 4.33, 4.37, 4.5 , 4.52, 4.55, 4.58, 4.61, 4.65, 4.68, 4.7 ,
       4.78, 4.8 , 4.82, 4.84, 4.85, 4.87, 4.9 , 4.95, 5.  , 5.1 , 5.2 ,
       5.3 , 5.31, 5.5 , 5.54, 5.6 , 5.8 , 5.83, 5.95, 6.  , 6.07, 6.18,
       6.25, 6.3 , 6.39, 6.5 , 6.9 , 6.92, 7.  , 7.14, 7.15, 7.25, 7.5 ,
       7.72, 8.  , 8.1 , 8.17, 8.25, 8.5 , 8.75, 9.  , 9.5 , 9.53, 9.8 ,
       9.98])
```

1-(b). K-Nearest Neighbors

Conducting the estimation without cross-validation first

In [32]:

```
knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn.fit(X_train, y_train)
```

Out[32]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='euclidean',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform')
```

In [33]:

```
print(classification_report(knn.predict(X_train), y_train))
```

```
              precision    recall  f1-score   support

           0       0.61      0.68      0.64       190
           1       0.79      0.73      0.76       310

   micro avg       0.71      0.71      0.71       500
   macro avg       0.70      0.71      0.70       500
weighted avg       0.72      0.71      0.71       500
```

Conducting the leave-one-out cross-validation (LOOCV) to find the best k

In [37]:

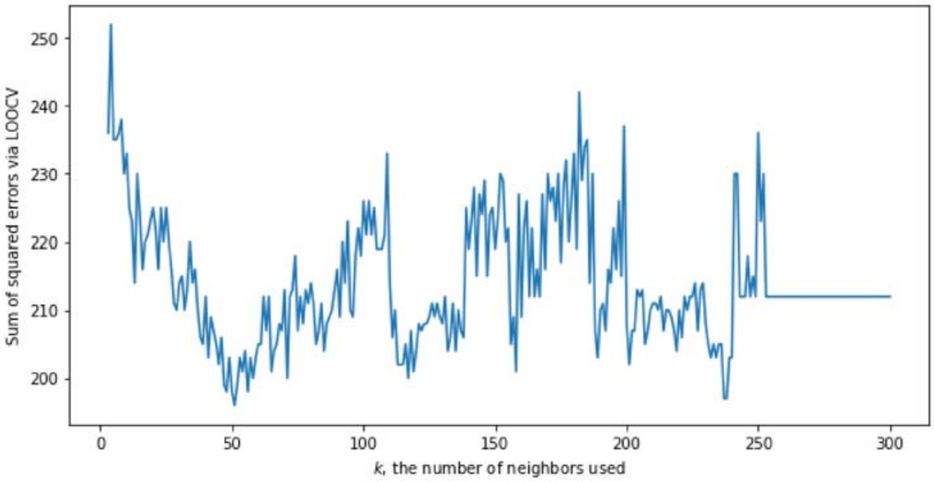
```
mse_vec = [] ## misnomer!
best_neighbor = 1
best_sse = np.inf
for neighbor in range(3, 301):
    knn = KNeighborsClassifier(n_neighbors=neighbor, metric='euclidean')
    challenger_sse = 0
    for train_index, test_index in loocv.split(X_train):
        y_this, X_this = y_train[train_index], X_train[train_index]
        knn.fit(X_this, y_this)
        ypred = knn.predict(X_train[test_index])
        diff = y_train[test_index] - ypred
        diff = diff ** 2
        challenger_sse += diff
    mse_vec.append(challenger_sse)
    if challenger_sse < best_sse:
        best_neighbor = neighbor
        best_sse = challenger_sse

print("Best k by LOOCV is {}.".format(best_neighbor))
print("The SSE for the best k is {}.".format(best_sse))
```

```
Best k by LOOCV is 51.
The SSE for the best k is [196].
```

In [39]:

```
cleanup_sse_vec = []
for i in mse_vec:
    sse = int(i[0])
    cleanup_sse_vec.append(sse)
```

In [47]:
```
plt.figure(figsize=(10, 5))
plt.plot(list(range(3, 301)), cleanup_sse_vec)
plt.ylabel("Sum of squared errors via LOOCV")
plt.xlabel(r"$k$, the number of neighbors used")
plt.show()
```



1-(c). Series

In [120]:
```
## creating interaction terms
for i in range(0, howmany):
    for j in range(i+1, howmany):
        inter = X_train[:, i] * X_train[:, j]
        if i == 0 and j == 1:
            stackonthis = inter.copy()
        else:
            stackonthis = \
                np.vstack((stackonthis, inter))

interactions = stackonthis.T
interactions.shape
```

Out[120]:
```
(500, 21)
```

In [121]:
```
## creating power terms
pwr = []
for i in range(0, howmany):
    lst = []
    for j in range(1, 11):
        if j == 1:
            appendthis = X_train[:, i].copy()
        else:
            appendthis = np.vstack((appendthis, X_train[:, i] ** j))
        lst.append(appendthis.T)
    pwr.append(lst)
```

In [122]:
```
## creating indices
import itertools
x = list(range(1, 11))
cases = [p for p in itertools.product(x, repeat=7)]
len(cases)
```

Out[122]:
```
10000000
```

Let's do random search; then try to find the neighborhood in which we might want to do grid search.

In [140]:
```
## 300 random draws
seed(60637)
indices_rando = randint(low=0, high=len(cases), size=300)
cases_rando = []
for i in indices_rando:
    cases_rando.append(cases[i])
```

```
In [376]: Xnames
```

```
Out[376]: ['kidslt6', 'kidsge6', 'age', 'educ', 'repwage', 'faminc', 'exper']
```

Here we do the LOOCV for the randomly selected cases.

```
In [141]: best_sse = np.inf
          best_case = 'what'
          sse_vec = []
          case_vec = []

          for case in cases_rando:
              for i, ca in enumerate(case):
                  whichone = pwr[i][ca-1].copy()
                  if ca == 1:
                      whichone = whichone.reshape((500, 1))

                  if i == 0:
                      new_X = whichone.copy()
                  else:
                      new_X = np.hstack((new_X, whichone))

              for j in range(0, 2):
                  if j == 1:
                      new_X = np.hstack((new_X, interactions))

                  challenger_sse = 0
                  for train_index, test_index in loocv.split(new_X):
                      X_, y_ = new_X[train_index], np.ravel(y_train[train_index])
                      reg.fit(X_, y_)
                      predy = list(reg.predict(new_X[test_index]))[0]
                      realy = list(y_train[test_index])[0]
                      sse_entry = (predy - realy) ** 2
                      challenger_sse += sse_entry

                  sse_vec.append(challenger_sse)
                  case_vec.append((case, j))

                  if challenger_sse < best_sse:
                      best_sse = challenger_sse
                      best_case = (case, j)
```

```
In [142]: print("Best case:", best_case)
          print("Best sse:", best_sse)
```

```
Best case: ((1, 1, 3, 3, 10, 1, 5), 1)
Best sse: 55.38502021339659
```

Doing grid search

```
In [148]: x = list(range(1, 3))
          gridcases = [p for p in itertools.product(x, repeat=7)]
          cleanup_gridcases = []
          for i, grid in enumerate(gridcases):
              newshit = []
              for j, hey in enumerate(grid):
                  enterthis = hey
                  if j == 2 or j == 3:
                      enterthis += 2
                  elif j == 4:
                      enterthis += 8
                  elif j == 6:
                      enterthis += 4
                  newshit.append(enterthis)
              cleanup_gridcases.append(newshit)
```

In [157]:
```
best_sse = np.inf
best_case = 'what'
best_X = 'what'
sse_vec = []
case_vec = []

for case in cleanup_gridcases:
    for i, ca in enumerate(case):
        whichone = pwr[i][ca-1].copy()
        if ca == 1:
            whichone = whichone.reshape((500, 1))

        if i == 0:
            new_X = whichone.copy()
        else:
            new_X = np.hstack((new_X, whichone))

    for j in range(0, 2):
        if j == 1:
            new_X = np.hstack((new_X, interactions))

        challenger_sse = 0
        for train_index, test_index in loocv.split(new_X):
            X_, y_ = new_X[train_index], np.ravel(y_train[train_index])
            reg.fit(X_, y_)
            predy = list(reg.predict(new_X[test_index]))[0]
            realy = list(y_train[test_index])[0]
            sse_entry = (predy - realy) ** 2
            challenger_sse += sse_entry

        sse_vec.append(challenger_sse)
        case_vec.append((case, j))

        if challenger_sse < best_sse:
            best_sse = challenger_sse
            best_case = (case, j)
            best_X = new_X.copy()
```

In [158]:
```
print("Best case:", best_case)
print("Best sse:", best_sse)
```

```
Best case: ([1, 1, 3, 3, 9, 1, 5], 1)
Best sse: 55.308385046039106
```

In [159]:
```
reg.fit(best_X, y_train)
newpredy = np.ravel(reg.predict(best_X))

under_0_or_over_1 = 0
for i in newpredy:
    if i < 0 or i > 1:
        under_0_or_over_1 += 1

print(under_0_or_over_1)
```

```
85
```

1-(d). Simple Probit

Checking unique values for $kidslt6$ and $kidsge6$.

In [171]:
```
print(np.unique(X[:, 0])) ## kidslt6
print(np.unique(X[:, 1])) ## kidsge6
```

```
[0. 1. 2. 3.]
[0. 1. 2. 3. 4. 5. 6. 7. 8.]
```

Checking unique values for the training set, just in case

```
In [220]: print(np.unique(X_train[:, 0]))
          print(np.unique(X_train[:, 1])) ## missing 6 and 7

          [0. 1. 2. 3.]
          [0. 1. 2. 3. 4. 5. 8.]
```

Creating dummies

```
In [221]: kidslt6_dummies = np.array(pd.get_dummies(X_train[:, 0]))
          kidsge6_dummies = np.array(pd.get_dummies(X_train[:, 1]))
```

```
In [222]: ## kidslt6 dummies: baseline 0
          kidslt6_dummies = kidslt6_dummies[:, 1:]
          print(kidslt6_dummies.shape)

          ## same for kidsge6 dummies: baseline 0
          kidsge6_dummies = kidsge6_dummies[:, 1:]
          print(kidsge6_dummies.shape)

          (500, 3)
          (500, 6)
```

Creating the "new" training dataset with the dummies

```
In [223]: newX_tr = np.hstack((X_train[:, 2:], kidsge6_dummies, kidslt6_dummies))
```

No "tuning" per se required, so no LOOCV and let us directly fit the model.

```
In [224]: probitfit = sts.Probit(endog=y_train.reshape((500, 1)), exog=newX_tr)
```

```
In [379]: A = probitfit.fit(maxiter=200)

          Warning: Maximum number of iterations has been exceeded.
                   Current function value: 0.323360
                   Iterations: 200
```

```
In [227]: predy_probit = A.predict()
```

```
In [228]: sse_probit = ((predy_probit - y_train) ** 2).sum()
```

```
In [229]: sse_probit

Out[229]: 51.02262631387188
```

1-(e).

```
In [424]: ## 0th case
          print(((y_train.mean() - y_test) ** 2).sum()/253)
          print(((y_train.mean() - y_test) ** 2).std()/253)
          print(((1 - y_test) ** 2).sum()/253)
          print(((1 - y_test) ** 2).std()/253)

          0.2476653280632411
          0.0002986797541466453
          0.44664031620553357
          0.0019649983825437206
```

```
In [423]: y_train.mean() >= 0.5

Out[423]: True
```

Kernel

```
In [500]: # [3.0, 4.0, 30.0, 12.0, 4.99, 94500.0, 45.0]
          coords, cell_y, sse = gauss_ke([1, 2, 1, 1, 2, 1, 1], X_df, y_df)
```

```
In [512]: X_test_coords = []
          for row in range(X_test.shape[0]):
              aaaa = X_test[row, :]
              row_coord = []
              for i, case in enumerate(aaaa):
                  if i in [0, 2, 3, 5, 6]:
                      row_coord.append(1)
                  elif i == 1:
                      if case <= 4:
                          row_coord.append(1)
                      else:
                          row_coord.append(2)
                  elif i == 4:
                      if case <= 4.99:
                          row_coord.append(1)
                      else:
                          row_coord.append(2)
              X_test_coords.append(row_coord)
```

```
In [513]: cases = []
          for i in X_test_coords:
              tup = tuple(i)
              for j, coord in enumerate(coords):
                  if tup == coord:
                      cases.append(j)
                      break
```

```
In [516]: hello = []
          for case in cases:
              hello.append(cell_y[case])
```

```
In [518]: hello = np.array(hello)
```

```
In [519]: ((hello - y_test) ** 2).sum()

Out[519]: 58.75195009649621
```

```
In [520]: print(((hello - y_test) ** 2).sum()/253)
          print(((hello - y_test) ** 2).std()/253)
          print((((hello >= 0.5) - y_test) ** 2).sum()/253)
          print((((hello >= 0.5) - y_test) ** 2).std()/253)

          0.23222114662646723
          0.0004999811911527698
          0.44664031620553357
          0.0019649983825437206
```

KNN

```
In [426]: knn = KNeighborsClassifier(n_neighbors=51)
          knn.fit(X_train, y_train)
          y_pred_proba_knn = knn.predict_proba(X_test)
          y_pred_knn = knn.predict(X_test)
```

```
In [428]: print(((y_pred_proba_knn[:, 1] - y_test) ** 2).sum()/253)
          print(((y_pred_proba_knn[:, 1] - y_test) ** 2).std()/253)
          print(((y_pred_knn - y_test) ** 2).sum()/253)
          print(((y_pred_knn - y_test) ** 2).std()/253)

          0.25047678530452716
          0.0005201877730300297
          0.44664031620553357
          0.0019649983825437206
```

Series

In [469]:
```
## creating interaction terms
for i in range(0, howmany):
    for j in range(i+1, howmany):
        inter = X_test[:, i] * X_test[:, j]
        if i == 0 and j == 1:
            stackonthis = inter.copy()
        else:
            stackonthis = \
                np.vstack((stackonthis, inter))

interactions_te = stackonthis.T
interactions_te.shape
```

Out[469]:
```
(253, 21)
```

In [476]:
```
for i, pwr in enumerate([1, 1, 3, 3, 9, 1, 5]):
    what = X_test[:, i].reshape((253, 1))
    if pwr > 1:
        for j in range(1, pwr+1):
            if j == 1:
                appendthis = what.copy()
            else:
                what_ = what ** j
                appendthis = np.hstack((appendthis, what_))
    else:
        appendthis = what.copy()
    print(appendthis.shape)

    if i == 0:
        appendhere = appendthis.copy()
    else:
        appendhere = np.hstack((appendhere, appendthis))
```

```
(253, 1)
(253, 1)
(253, 3)
(253, 3)
(253, 9)
(253, 1)
(253, 5)
```

In [477]:
```
appendhere.shape
```

Out[477]:
```
(253, 23)
```

In [478]:
```
newXte = np.hstack((appendhere, interactions_te))
```

In [479]:
```
series_y_te = np.ravel(reg.predict(newXte))
```

In [480]:
```
series_bayes = series_y_te >= 0.5
```

In [481]:
```
print(((series_y_te - y_test) ** 2).sum()/253)
print(((series_y_te - y_test) ** 2).std()/253)
print(((series_bayes - y_test) ** 2).sum()/253)
print(((series_bayes - y_test) ** 2).std()/253)
```

```
0.11921625655511615
0.0008750072473229852
0.14624505928853754
0.0013966471364106304
```

Probit

In [451]:
```
kidslt6_dummies2 = np.array(pd.get_dummies(X_test[:, 0]))
kidsge6_dummies2 = np.array(pd.get_dummies(X_test[:, 1]))
```

In [452]:
```
kidsge6_dummies2[:, -2] = kidsge6_dummies2[:, -1] + kidsge6_dummies2[:, -2]
kidsge6_dummies2 = kidsge6_dummies2[:, 0:7]
```

In [453]:
```
print(X_test[:, 2:].shape, kidsge6_dummies2.shape)
```

```
(253, 5) (253, 7)
```

In [461]:

```
newX_te = np.hstack((X_test[:, 2:], kidsge6_dummies2[:, 1:]))
newX_te = np.hstack((newX_te, kidslt6_dummies2[:, 1:]))
```

In [464]:

```
hey = A.predict(newX_te) >= 0.5
```

In [467]:

```
print(((A.predict(newX_te) - y_test) ** 2).sum()/253)
print(((A.predict(newX_te) - y_test) ** 2).std()/253)
print(((hey - y_test) ** 2).sum()/253)
print(((hey - y_test) ** 2).std()/253)
```

```
0.11595477182068964
0.0008378146031466489
0.15019762845849802
0.0014121147386744572
```