PS1_Answer

May 6, 2019

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib
        import matplotlib.pyplot as plt
        from pandas.plotting import scatter_matrix
        from scipy.stats import gaussian_kde
        import warnings
        warnings.filterwarnings("ignore")
```

0.1 Question 1

0.1.1 1(a)

```
In [2]: bq_data = np.loadtxt('data/BQmat_orig.txt', delimiter=',')
        #bq_data[43][5]
```

```
In [3]: from mpl_toolkits.mplot3d import Axes3D
        %matplotlib notebook
        prcntl = np.array([0.25,0.25,0.2,0.1,0.1,0.09,0.01])
        prcntl_mdpts = np.array([0.125, 0.375, 0.60, 0.75, 0.85, 0.94, 0.995])
        age_vec = np.arange(18,96)
        income_mat, age_mat = np.meshgrid(prcntl_mdpts, age_vec)
        fig = plt.figure()
        ax = fig.gca(projection='3d')
        ax.plot_surface(age_mat, income_mat, bq_data)
        ax.set_title('Raw distribution of bequest recipient proportion')
        ax.set_xlabel('Age')
        ax.set_ylabel('Ability Types')
        ax.set_zlabel('Percent of BQ received')
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
Out[3]: Text(0.5,0,'Percent of BQ received')
```

0.1.2 1(b)

```
In [4]: bq_data = np.loadtxt('data/BQmat_orig.txt', delimiter=',')
        ages_vec = np.arange(18, 96)
        abils_midpt = np.array([0.125, 0.375, 0.60, 0.75, 0.85, 0.94, 0.995])
        prop_mat_inc = np.sum(bq_data, axis=0)
        prop_mat_age = np.sum(bq_data, axis=1)
        N_samp = 70000
        age_probs = np.random.multinomial(N_samp, prop_mat_age)
        income_probs = np.random.multinomial(N_samp, prop_mat_inc)
        age_freq = np.array([])
        inc_freq = np.array([])

        # creating a distribution of age values
        for age, num_s in zip(ages_vec, age_probs):
            vec_age_s = np.ones(num_s)
            vec_age_s *= age
            age_freq = np.append(age_freq, vec_age_s)

        # creating a distribution of ability type values
        for abil, num_j in zip(abils_midpt, income_probs):
            vec_abil_j = np.ones(num_j)
            vec_abil_j *= abil
            inc_freq = np.append(inc_freq, vec_abil_j)

        bandwidth = 0.1
        data = np.vstack((age_freq, inc_freq))
        density = gaussian_kde(data, bw_method=bandwidth)
        data.shape

Out[4]: (2, 70000)

In [5]: coords = np.vstack([item.ravel() for item in [age_mat, income_mat]])
        BQkde = density(coords).reshape(age_mat.shape)
        BQkde_scaled = BQkde / BQkde.sum()

In [6]: fig = plt.figure()
        ax = fig.gca(projection='3d')
        ax.plot_surface(age_mat, income_mat, BQkde_scaled)
        ax.set_title('KDE estimated distribution of bequest recipient proportion')
        ax.set_xlabel('Age')
        ax.set_ylabel('Ability Types')
        ax.set_zlabel('Percent of BQ received')

<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>
```

```
Out[6]: Text(0.5,0,'Percent of BQ received')
```

Choice of $\lambda = 0.1$, since it generates a smooth surface, and is closest to the real data.

```
In [7]: print("Estimated density for bequest recipients who are age 61 in the 6th lifetime inc

Estimated density for bequest recipients who are age 61 in the 6th lifetime income category =
```

0.2 Question 2

0.2.1 2(a)

```
In [8]: df = pd.read_csv('data/Auto.csv', na_values='?')
        df.dropna(inplace=True)
        df.columns=['mpg', 'cyl', 'displ', 'hpwr', 'wgt', 'accl', 'yr', 'orgn','name']
        df.head()
```

Out[8]:

	mpg	cyl	displ	hpwr	wgt	accl	yr	orgn	name
0	18.0	8	307.0	130.0	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	1	ford torino

```
In [9]: df['mpg_high'] = 0
        df.mpg_high[df.mpg>=df.mpg.median()] = 1
        df['orgn1'] = (df.orgn==1).astype(int)
        df['orgn2'] = (df.orgn==2).astype(int)
        df.head()
```

Out[9]:

	mpg	cyl	displ	hpwr	wgt	accl	yr	orgn	name	\
0	18.0	8	307.0	130.0	3504	12.0	70	1	chevrolet chevelle malibu	
1	15.0	8	350.0	165.0	3693	11.5	70	1	buick skylark 320	
2	18.0	8	318.0	150.0	3436	11.0	70	1	plymouth satellite	
3	16.0	8	304.0	150.0	3433	12.0	70	1	amc rebel sst	
4	17.0	8	302.0	140.0	3449	10.5	70	1	ford torino	

	mpg_high	orgn1	orgn2
0	0	1	0
1	0	1	0
2	0	1	0
3	0	1	0
4	0	1	0

```
In [10]: from sklearn.linear_model import LogisticRegression
         from sklearn.model_selection import train_test_split
         import graphviz
         from dask import delayed
```

```
from dask import compute
import dask.array as da
from statistics import mean
```

In [11]:
```
%%time
N_bs = 100

err_vec = np.zeros(N_bs)
X = df[['cyl', 'displ', 'hpwr', 'wgt', 'accl', 'yr', 'orgn1', 'orgn2']].values
y = df.mpg_high.values

for bs_ind in range(N_bs):
    X_train, X_test, y_train, y_test = \
            train_test_split(X, y, test_size=0.35, random_state=1000+bs_ind)
    LogReg = LogisticRegression(n_jobs = None)
    LogReg.fit(X_train, y_train)
    y_pred = LogReg.predict(X_test)
    err_vec[bs_ind] = (y_test == y_pred).mean()

mean_err = err_vec.mean()

print('The average error rate =', mean_err)
```

```
The average error rate = 0.9009420289855073
CPU times: user 125 ms, sys: 4.03 ms, total: 129 ms
Wall time: 134 ms
```

0.2.2 2(b)

In [12]:
```
def paral_bs(bs_ind, seed, X, y):
    X_train, X_test, y_train, y_test = \
            train_test_split(X, y, test_size=0.35, random_state=seed+bs_ind)
    LogReg = LogisticRegression(n_jobs = None)
    LogReg.fit(X_train, y_train)
    y_pred = LogReg.predict(X_test)
    return (y_test == y_pred).mean()
```

In [13]:
```
%%time
N_bs = 100
err_vec = []
X = df[['cyl', 'displ', 'hpwr', 'wgt', 'accl', 'yr', 'orgn1', 'orgn2']].values
y = df.mpg_high.values

for bs_ind in range(N_bs):
    sample_MSE = delayed(paral_bs(bs_ind, 1000, X, y))
    err_vec.append(sample_MSE)

mean_err = delayed(mean)(err_vec)
```

```
        mean_err = compute(mean_err)

        print('The average error rate =', mean_err[0])
```

```
The average error rate = 0.9009420289855072
CPU times: user 155 ms, sys: 7.95 ms, total: 163 ms
Wall time: 175 ms
```