DATA 7202 Assignment 3 Report

Name: Yupeng Wu
No: 45960600

1.

In order to apply the Crude Monte Carlo algorithm, first we need get a random number $x \in [1, 8]$. Second, we need to compute the $f(x)$ value and take the product of $f(x)$ and the range of the integral as the estimate of the integral $est(x) = f(x) * range$, where range is 7. This is just one estimate based on a random picked x and we need more. Finally, after 10000 trials, we take the mean of the estimates as the result. Please check my code for question 1 in the appendix.

For the 95% confidence interval, this is

$$[mean(X) - 1.96\frac{std(X)}{\sqrt{N}}, mean(X) + 1.96\frac{std(X)}{\sqrt{N}}]$$

Where $X = \{est(x_1), estf(\ x_2), \dots, estf(x_N)\}, N = 10000.$

The result changes each time you run. One of my result is 234.07 with the confidence interval of $[217.40, 250.73]$. Compared with the ground, which is 235.26, the result is pretty close to it and the true value is inside the confidence interval.

2.

This method might cause data leakage when we are doing some pre-processing outside the cross-validation algorithm. Using high correlated features to train the model, we may overfit the model and the K-Fold cross-validation accuracy is an overestimate of the true test error. In this case, this is not a good method to show the inner-connection and I will not obtain the prediction error.

3.

Note: Please check my code for question 3 in the appendix.

(a)

After using the “LabelEncode” function, the categorical values changed to int32.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 263 entries, 0 to 262
Data columns (total 20 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   AtBat      263 non-null    int64
 1   Hits       263 non-null    int64
 2   HmRun      263 non-null    int64
 3   Runs       263 non-null    int64
 4   RBI        263 non-null    int64
 5   Walks      263 non-null    int64
 6   Years      263 non-null    int64
 7   CAtBat     263 non-null    int64
 8   CHits      263 non-null    int64
 9   CHmRun     263 non-null    int64
 10  CRuns      263 non-null    int64
 11  CRBI       263 non-null    int64
 12  CWalks     263 non-null    int64
 13  League     263 non-null    int32
 14  Division   263 non-null    int32
 15  PutOuts    263 non-null    int64
 16  Assists    263 non-null    int64
 17  Errors     263 non-null    int64
 18  Salary     263 non-null    float64
 19  NewLeague  263 non-null    int32
dtypes: float64(1), int32(3), int64(16)
memory usage: 38.1 KB
```

```
df.head()
```

CHmRun	CRuns	CRBI	CWalks	League	Division	PutOuts	Assists	Errors	Salary	NewLeague
69	321	414	375	1	1	632	43	10	475.0	1
63	224	266	263	0	1	880	82	14	480.0	0
225	828	838	354	1	0	200	11	3	500.0	1
12	48	46	33	1	0	805	40	4	91.5	1
19	501	336	194	0	1	282	421	25	750.0	0

(b)

10-Fold cross-validation mean squared error: 116599.01.

	1	2	3	4	5
MSE	109666.78	37953.39	156572.06	82386.08	84002.25
	6	7	8	9	10
MSE	67162.76	270025.66	135034.87	132156.31	91029.98

(c)

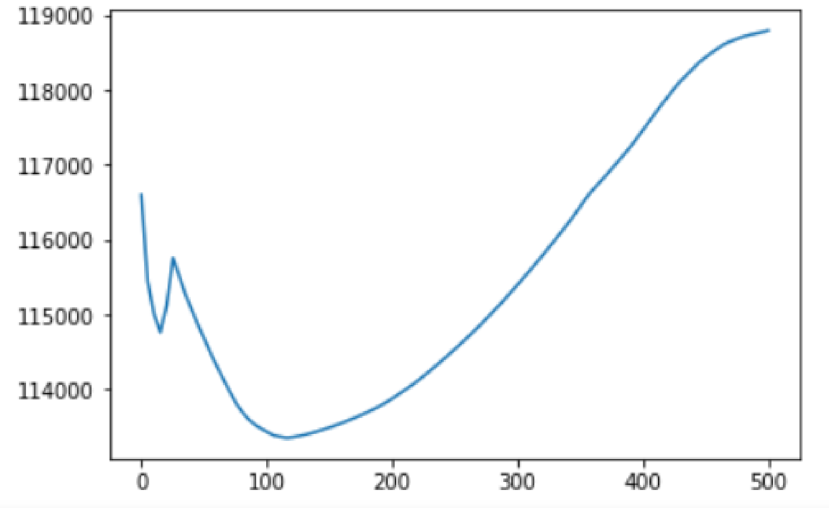
The above picture shows the MSE of using different components, ranges from 1 component to 19 components.

The optimal choice is the regression with 18 components. It has the minimum 10-Fold cross-validation mean squared error, which is 115083.91.

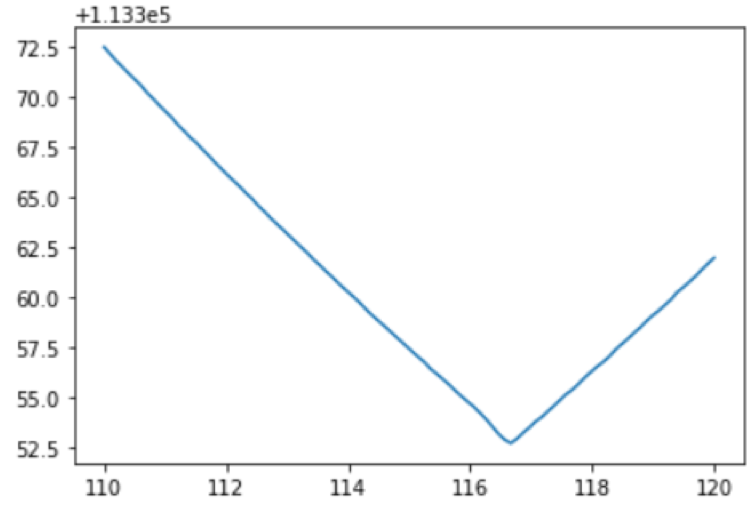
Besides, there is a huge decrease when there are 6 components in the model and the mean squared error is 118508.16. This shows that the 6^{th} component has a great effect on the model. If you do not want to have so much components in the model, this is also an acceptable option to me.

(d)

Using Lasso function in the sklearn.linear_model to build a lasso model. In order to find the minimum mean squared error of the 10-Fold cross-validation, I set 100 points range from 0 to 500.



From the above figure we can see that the minimum value is located near the 100. Then I set 100 points from 110 to 120 for a more accurate result.



From the more accurate figure and smaller step, we can get the best $\lambda \approx 116.6$ with the mean squared error $mse_{116} \approx 113354.19$.

4.

Original thinking:

① Generate U from $U(0,1)$.

② Determine the integer index $I \in [0, n]$ such that

$$\frac{I}{n+1} < U \leq \frac{I+1}{n+1}$$

Then the random variable will be $\frac{I+1}{n+1}$.

Advanced method:

① Generate U from $U(0,1)$.

② Set the random variable $x = \frac{floor(U*(n+1))}{n+1}, x \in (0,1)$. Here the function $floor(x)$ means to get the greatest integer less than or equal to x.

Time complexity:

The original thinking is $O(ln(n))$ because the second step requires a search that starts with $I = 0$ and keep adding 1 to I until there is a I that $U \leq \frac{I+1}{n+1}$.

But the advanced method only needs $O(1)$, since it just need one generation from $U(0,1)$ and a return function.

5.

(a)

$$Var(Z) = E(Z^2) - (E(Z)^2)$$

Since $Z_i = 1_{\{X_i \geq \gamma\}}$, then

$$E(Z^2) = E(Z) = l_\gamma$$

Do the math:

$$CV^2 = \frac{Var(Z)}{E(Z)^2} = \frac{E(Z^2) - (E(Z)^2)}{E(Z)^2} = \frac{l_\gamma - l_\gamma^2}{l_\gamma} = \frac{1}{l_\gamma} - 1$$

(b)

Set the relative error of the estimator is ε.

$$\varepsilon = \frac{\sqrt{Var(\widehat{l_\gamma})}}{l_\gamma} = \frac{\sqrt{l_\gamma - l_\gamma^2}}{\sqrt{N l_\gamma^2}} = \sqrt{\frac{1 - l_\gamma}{N l_\gamma}}$$

(c)

$$lim \frac{ln\, E(Z^2)}{ln(E(Z)^2)} = lim \frac{ln\, l_\gamma}{ln\, l_\gamma^2} = \frac{1}{2}$$

The result is not 1, which proves that the CMC estimator is not logarithmically efficient.

Appendix

Q1:

In [1]:

```
import numpy as np
import math
import random
from matplotlib import pyplot as plt
from IPython.display import clear_output
```

In [2]:

```
def get_rand_number(min_value, max_value):
    range = max_value - min_value
    choice = random.uniform(0,1)
    return min_value + range*choice
```

In [3]:

```
def f(x):
    return (3*x+x**2-200*math.cos(x))
```

In [4]:

```
def crude_monte_carlo(num_samples=10000):
    lower_bound = 1
    upper_bound = 8

    sum_of_samples = 0
    X = []
    for i in range(num_samples):
        x = get_rand_number(lower_bound, upper_bound)
        sum_of_samples += f(x)
        X.append(f(x)*7)

    monte_carlo = (upper_bound - lower_bound) * float(sum_of_samples/num_samples)

    return monte_carlo, X
```

In [5]:

```
groud_truth = 235.26
MC_score, X = crude_monte_carlo()
low = np.mean(X)-1.96*np.std(X)/math.sqrt(10000)
up = np.mean(X)+1.96*np.std(X)/math.sqrt(10000)
print('Result is:', MC_score, '\nWith confidence interval [', low, ',', up, ']')
```

```
Result is: 240.27140391531992
With confidence interval [ 223.57334561106956 , 256.9694622195686 ]
```

Q3:

(a)

In [6]:

```
import pandas as pd
from sklearn import preprocessing
```

In [7]:

```
def LabelEncode(data):
    le = preprocessing.LabelEncoder()
    le.fit(data)
    return le.transform(data)
```

In [8]:

```
df = pd.read_csv('Hitters.csv')
df['League'] = LabelEncode(df['League'])
df['Division'] = LabelEncode(df['Division'])
df['NewLeague'] = LabelEncode(df['NewLeague'])
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 263 entries, 0 to 262
Data columns (total 20 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   AtBat      263 non-null    int64
 1   Hits       263 non-null    int64
 2   HmRun      263 non-null    int64
 3   Runs       263 non-null    int64
 4   RBI        263 non-null    int64
 5   Walks      263 non-null    int64
 6   Years      263 non-null    int64
 7   CAtBat     263 non-null    int64
 8   CHits      263 non-null    int64
 9   CHmRun     263 non-null    int64
 10  CRuns      263 non-null    int64
 11  CRBI       263 non-null    int64
 12  CWalks     263 non-null    int64
 13  League     263 non-null    int32
 14  Division   263 non-null    int32
 15  PutOuts    263 non-null    int64
 16  Assists    263 non-null    int64
 17  Errors     263 non-null    int64
 18  Salary     263 non-null    float64
 19  NewLeague  263 non-null    int32
dtypes: float64(1), int32(3), int64(16)
memory usage: 38.1 KB
```

(b)

In [9]:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
y = df.iloc[:, -2]
X = df.drop(columns=['Salary'])
lm = LinearRegression().fit(X, y)
```

In [10]:

```
score = -1*cross_val_score(lm, X, y, cv=10, scoring='neg_mean_squared_error')
score.mean()
```

Out[10]:

116599.01367380263

(c)

In [11]:

```
from sklearn.decomposition import PCA
from sklearn import preprocessing
import matplotlib.pyplot as plt

df_scale = pd.read_csv('Hitters.csv')
df_scale['League'] = LabelEncode(df['League'])
df_scale['Division'] = LabelEncode(df['Division'])
df_scale['NewLeague'] = LabelEncode(df['NewLeague'])

y = df_scale['Salary']
X_scale = df_scale.drop(['Salary'], axis=1)

X_scale
```

Out[11]:

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI
0	315	81	7	24	38	39	14	3449	835	69	321	414
1	479	130	18	66	72	76	3	1624	457	63	224	266
2	496	141	20	65	78	37	11	5628	1575	225	828	838
3	321	87	10	39	42	30	2	396	101	12	48	46
4	594	169	4	74	51	35	11	4408	1133	19	501	336
...	...	...	...	...	...	...	...	...	...	...	...	...
258	497	127	7	65	48	37	5	2703	806	32	379	311
259	492	136	5	76	50	94	12	5511	1511	39	897	451
260	475	126	3	61	43	52	6	1700	433	7	217	93
261	573	144	9	85	60	78	8	3198	857	97	470	420
262	631	170	9	77	44	31	11	4908	1457	30	775	357

263 rows × 19 columns

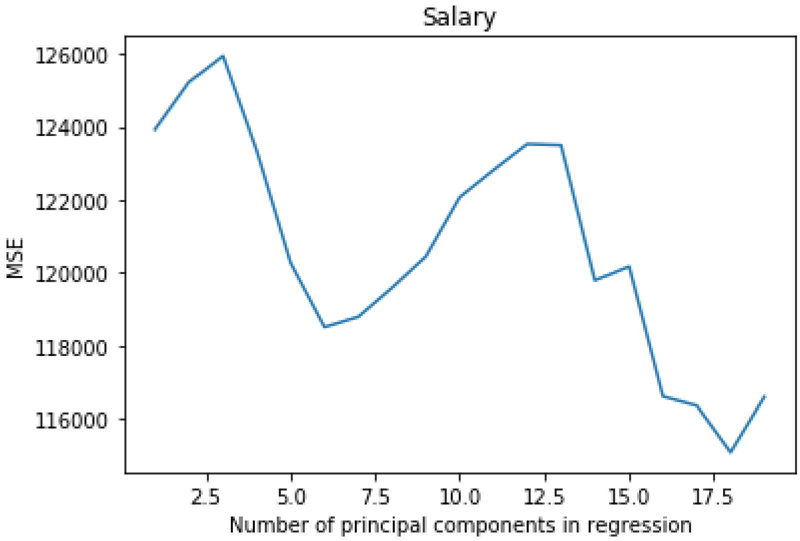
In [12]:

```
from sklearn.preprocessing import scale
pca = PCA()
X_scale = pca.fit_transform(scale(X_scale))
```

In [13]:

```
n = len(X_scale)

lm_scale = LinearRegression()
mse = []

score = -1*cross_val_score(lm_scale, X_scale, y, cv=10, scoring='neg_mean_squared_error').mean()

for i in range(1, 20):
    score = -1*cross_val_score(lm_scale, X_scale[:,:i], y.ravel(), cv=10, scoring='neg_mean_squa
red_error').mean()
    mse.append(score)
    print('MSE with', i, 'components:', score)

plt.plot(range(1,20), mse)
plt.xlabel('Number of principal components in regression')
plt.ylabel('MSE')
plt.title('Salary');
```

```
MSE with 1 components: 123924.13089980235
MSE with 2 components: 125219.8186020469
MSE with 3 components: 125936.4971858609
MSE with 4 components: 123338.38059497121
MSE with 5 components: 120267.60729131899
MSE with 6 components: 118508.15848939032
MSE with 7 components: 118789.33335711618
MSE with 8 components: 119588.5389074013
MSE with 9 components: 120446.2134914831
MSE with 10 components: 122072.10474467053
MSE with 11 components: 122818.16267921466
MSE with 12 components: 123533.98633524492
MSE with 13 components: 123501.9866236398
MSE with 14 components: 119791.3805164588
MSE with 15 components: 120171.99674286325
MSE with 16 components: 116609.51312630429
MSE with 17 components: 116359.68557326547
MSE with 18 components: 115083.91154069177
MSE with 19 components: 116599.0136738026
```



(d)

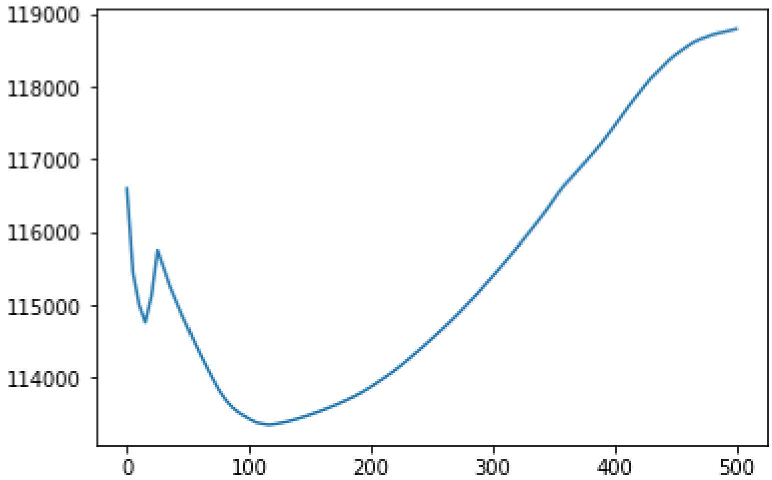
In [14]:

```
from sklearn import linear_model
alphas = np.linspace(0, 500, num=100)

lasso_mse = []
for alpha in alphas:
    lasso = linear_model.Lasso(alpha=alpha, max_iter=10000000)
    score = -1*cross_val_score(lasso, X, y, cv=10, scoring='neg_mean_squared_error').mean()
    lasso_mse.append(score)

plt.plot(alphas, lasso_mse)
```

```
t.py:476: UserWarning: Coordinate descent with no regularization may lead to unexp
ected results and is discouraged.
  positive)
C:\Users\simon\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descen
t.py:476: ConvergenceWarning: Objective did not converge. You might want to increa
se the number of iterations. Duality gap: 11045094.301265078, tolerance: 4912.8603
55045331
  positive)
```

Out[14]:

```
[<matplotlib.lines.Line2D at 0x1a31b524648>]
```



In [15]:

```python
from sklearn import linear_model
alphas = np.linspace(110, 120, num=100)
```

In [16]:

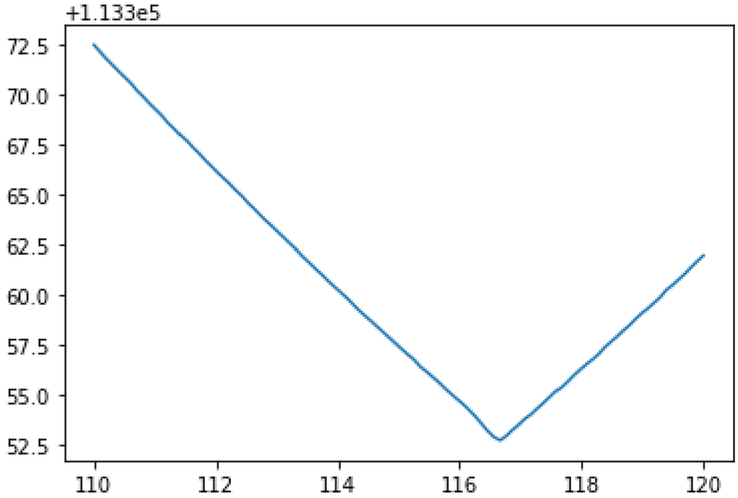
```python
lasso_mse = []
for alpha in alphas:
    lasso = linear_model.Lasso(alpha=alpha, max_iter=10000000)
    score = -1*cross_val_score(lasso, X, y, cv=10, scoring='neg_mean_squared_error').mean()
    lasso_mse.append(score)
```

In [17]:

```
plt.plot(alphas, lasso_mse)
```

Out[17]:

```
[<matplotlib.lines.Line2D at 0x1a31b592648>]
```



In [18]:

```
print('The best lambda is:', alphas[lasso_mse.index(min(lasso_mse))])
```

```
The best lambda is: 116.66666666666667
```

In [19]:

```
lasso = linear_model.Lasso(alpha=alphas[lasso_mse.index(min(lasso_mse))], max_iter=10000000)
print('The MSE is:', -1*cross_val_score(lasso, X, y, cv=10, scoring='neg_mean_squared_error').me
an())
```

```
The MSE is: 113352.68244494035
```

In [ ]:

```

```