Worksheet

Stochastic Models and Markov Chains

I. Poisson Process question

A stochastic process is defined as a collection of random variables indexed by time. The Poisson process is one such example of discrete stochastic process. The following Code and questions are based on a poisson process.

A supplier of aircraft flight control system spares wants to find the inventory of spares that would make sure that the demand for spares over the next two years is met in at least 90% of the cases (called fill rate)

He has the following data : Hydraulic_pumps

Code:

```
import numpy as np
import pandas as pd
from scipy.stats import poisson

def get_k(mu,fr):
    cumulative = 0
    k = 0
    while(cumulative<fr):
        cumulative = poisson.cdf(mu=mu,k=k)
        k+=1
    return k-1

path = 'path/to/data/'

data = pd.read_csv(path+'Hydraulic_pumps.csv')

# Converting df to np array
X = data.to_numpy()

# Maximum Likelihood Estimate of the λ parameter for a poisson
distribution is
# λ_mle = 1/((1/n) * Σ Xi)
MLE_lambda = 1/(np.sum(X)/len(X))
num_years = 2

# Expected demand in 2 years
expected_demand = MLE_lambda*num_years*365
fill_rate = 0.90
```

```
# We need to get the smallest k such that the cumulative distribution
value
# For the poisson distribution becomes greater than fill_rate
spare_inventory = get_k(expected_demand,fill_rate)
print(spare_inventory)
```

Questions:

1. Try the same question for different durations of time and different fill rates and observe how the value of k changes.
2. What other real-life examples of stochastic processes can you think of?

II. Markov Chains - Question-1

A basic code example to see how Markov chains work. There are three states here [‘Sleep’, ‘Ice Cream’, ‘Run’] with their transition probabilities defined.

Code:

```
import numpy as np
import random as rm

# Possible states
states = ["Sleep","Icecream","Run"]

# Possible sequences of events
transitionName = [["SS","SR","SI"],["RS","RR","RI"],["IS","IR","II"]]

# Probabilities matrix (transition matrix)
transitionMatrix = [[0.2,0.6,0.2],[0.1,0.6,0.3],[0.2,0.7,0.1]]

def activity_forecast(days, activityFirstDay):
      # Set the starting state
      activityToday = activityFirstDay
      activityList = [activityToday]
      i = 0
      prob = 1
      while i != days:
      # For the number of days, generate a new state and calculate
      probability of reaching that state
            if activityToday == states[0]:
                  change =
                  np.random.choice(transitionName[0],replace=True,p=trans
                  itionMatrix[0]) # Choose a next change

      # Calculate probability and  append state to activityList
      according to which state to change to
```

```
            if change == transitionName[0][0]:
                  prob = prob * transitionMatrix[0][0]
                  activityList.append(states[0])
            elif change == transitionName[0][1]:
                  prob = prob * transitionMatrix[0][1]
                  activityToday = states[1]
                  activityList.append(states[1])
            else:
                  prob = prob * transitionMatrix[0][2]
                  activityToday = states[2]
                  activityList.append(states[2])

#Repeat above step for all 3 different states
      elif activityToday == states[1]:
            change =
            np.random.choice(transitionName[1],replace=True,p=trans
            itionMatrix[1])
            if change == transitionName[1][1]:
                  prob = prob * transitionMatrix[1][1]
                  activityList.append(states[1])
            elif change == transitionName[1][0]:
                  prob = prob * transitionMatrix[1][0]
                  activityToday = states[0]
                  activityList.append(states[0])
            else:
                  prob = prob * transitionMatrix[1][2]
                  activityToday = states[2]
                  activityList.append(states[2])

      elif activityToday == states[2]:
            change =
            np.random.choice(transitionName[2],replace=True,p=trans
            itionMatrix[2])
            if change == transitionName[2][2]:
                  prob = prob * transitionMatrix[2][2]
                  activityList.append(states[2])
            elif change == transitionName[2][0]:
                  prob = prob * transitionMatrix[2][0]
                  activityToday = states[0]
                  activityList.append(states[0])
            else:
                  prob = prob * transitionMatrix[2][1]
                  activityToday = states[1]
                  activityList.append(states[1])
      i += 1
return activityList
```

```
if
sum(transitionMatrix[0])+sum(transitionMatrix[1])+sum(transitionMatrix[
1]) != 3:
     print("Probabilities of transition matrix do not add up to 1.
     Cannot execute. Exiting...")
else:

     # To save every activityList
     list_activity = []
     count = 0

     activityStart = np.random.choice(states)

     # Running it for 10000 iterations
     for iterations in range(10000):

               list_activity.append(activity_forecast(2,
          activityStart))

     activityEnd = np.random.choice(states)

     # Iterate through the list to get a count of all activities ending
     in state activityEnd
     for smaller_list in list_activity:
          if(smaller_list[2] == activityEnd):
               count += 1

     # Calculate the probability of starting from state activityStart
     and ending at state activityEnd

     percentage = (count/10000) * 100
     print("The probability of starting at state: " + activityStart + "
     and ending at state: " + activityEnd + " is " + str(percentage) +
     "%")
```

Questions:

1. Try printing activityList, what does it output?
2. Try with different states, transitions and probabilities. Be innovative!
3. What happens if there is an absorbing state in the matrix? Does the code still work? How does activityList change?

III. Markov Chains - Question 2

Let's try to use a Markov model to solve the real life application of text generation! We apply Markov Property to generate an essay by considering each word used in the given dataset of essays and for each word, create a dictionary of words that are used next.

The text file contains some popular essays by the computer scientist and essayist Paul Graham. Find the dataset here: Essays

Code:

```
# To generate text simulations

import numpy as np

# PART 1: SETTING UP THE MARKOV STATES

path = 'path/to/data'
essays = open(path+'/essays.txt', encoding='utf8').read()

# Split the data into individual words
corpus = essays.split()

# Generate different pairs of words in the corpus - save space by using
a generator object to generate every pair
def make_pairs(corpus):
     for i in range(len(corpus) - 1):
          yield (corpus[i], corpus[i + 1])

pairs = make_pairs(corpus)

# Initialise an empty dictionary to store the words
word_dict = {}

# There are two parts to this loop. For every pair of words, if the
first word in the pair is already a key in the dictionary, the next
word is added to the list of words that follow the first word.
# But if the word is not a key, then a new (key,value) entry is created
for that pair of words.

for word_1, word_2 in pairs:
     if word_1 in word_dict.keys():
          word_dict[word_1].append(word_2)
     else:
          word_dict[word_1] = [word_2]

# PART 2:- GENERATING TEXT USING MARKOV CHAIN

# Randomly pick the first word
first_word = np.random.choice(corpus)

# Pick the first word as a capitalized word so that the picked word is
not taken from in between a sentence
```

```
while first_word.islower():
      first_word = np.random.choice(corpus)

# Start the chain from the picked word
chain = [first_word]

# Initialize the number of stimulated words
n_words = 100

# Each word in the chain is randomly sampled from the list of words
which have followed that specific word to create a chain of length
n_words
for i in range(n_words):
      chain.append(np.random.choice(word_dict[chain[-1]]))

# Return the chain as a string
print(' '.join(chain), '.')
```

Questions:

1. Try changing the number of words generated. Till when does the output make sense? How can it be improved? (Hint: we have not used transition probabilities here!)
2. Try a different dataset to get outputs of sentence generation in different scenarios.
3. How would it work if we were to add transition probabilities to this? How would you calculate the probability of going to the next word? Would it give a better speech output? Explore and have fun! (Hint: transition probabilities here can be calculated by using frequency of word occurrence.)

Additional Reading Material

1. Good examples of Stochastic Processes
2. Basics of the Poisson Process
3. Basic Markov Notes
4. A great 2-state Markov chain simulation
5. MarkovChain package in R
6. Markov Chain Analysis - A good example in R
7. Comparative study between HMM and LSTM*
8. Markov chains with random transition matrices*

*Out of syllabus

Sparse PCA and Latent Semantic Analysis

I. Sparse PCA

Problem description: In the following dataset various features such as Age, Loan Type, Gender, Total Income and 54 other parameters are used to determine if a request for a loan has been granted or not. You must already know about the Curse of Dimensionality and the problems associated with having too many features. Here we use Sparse PCA to reduce the number of continuous-valued features.
Mortgage Dataset
Modified Mortgage Dataset

Code:

```
import numpy as np
import pandas as pd
from sklearn.decomposition import SparsePCA

path = 'path/to/your/data/file/'
#Use the modified dataset and not the original - it is there only for
your reference
data = pd.read_csv(path+'Mortgage_data.csv')

X_df = data.drop(['ID','Decision'],axis=1)
X = X_df.to_numpy()
print(X,X.shape)
transformer = SparsePCA(n_components=10, random_state=0, max_iter=500,
alpha = 1)
X_transformed = transformer.fit_transform(X)

print(X_transformed,X_transformed.shape)
#The following line prints [n_components,n_features] size numpy array
print(transformer.components_)
```

Questions:

1. Change the value of alpha in the function SparsePCA, how does it affect the result? Is it still sparse at alpha = 0?
2. Reduce the value of max_iter in the function SparsePCA. What effect does this have on the new components?
3. Why did we choose Sparse PCA here instead of PCA? What insight could you glean by making this decision?
4. Why did we remove the categorical variables from the data before performing SPCA?

II. Latent Semantic Analysis

Problem Description:- Topic modeling is analyzing text data to discover the abstract "topics" that occur in a collection of documents. One efficient way of doing this is by using LSA to capture the hidden context behind the words of a document and group them into clusters called “topics”. In this problem, the following collection of Medium articles are modeled to get topics that occur in the documents.

Link to the dataset:- Medium Articles Dataset

Code:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
import pandas as pd

# If you wish to run the code on a simpler dataset to test it out,
uncomment the below line and run fit_transform on 'documents'
# documents = ["Alpine snow winter boots", "Snow winter jacket",
"Alpine winter gloves", "Ice skating winter wear", "Ice cream in
summer", "Summer tank tops", "Beach footwear sandals"]

# Path to articles.csv
path = 'path/to/data'
data = pd.read_csv('articles.csv', engine='python', nrows=250) # Can
choose to read how many ever rows required

# Clean the data to remove everything except alphabets`
data.text = data.text.str.replace("[^a-zA-Z#]", " ")

# Create a document-term matrix
vectorizer = TfidfVectorizer(stop_words='english', # removes some
common stop words from data
                             max_df = 0.5, # ignore terms that have a
document frequency strictly higher than the given threshold
                              smooth_idf=True, use_idf = True) # to
prevent zero divisions

X = vectorizer.fit_transform(data.text) # fit vectorizer on 'text'
column of 'articles.csv'

# Using SVD, we represent each and every term and document as a vector
# Here, we use truncated SVD and generate only specified number of
eigenvectors that correspond to desired number of topics
svd_model = TruncatedSVD(n_components = 10, algorithm='randomized',
n_iter=100, random_state=122)
# Change n_components according to number of topics you want data to be
sorted into
svd_model.fit(X)
```

```
# Use the vectors generated by the svd to find similar words and
similar documents using the cosine similarity method
# Access the topics using svd_model.components_ to print a few most
important words in each of the topics

terms = vectorizer.get_feature_names()

for i, comp in enumerate(svd_model.components_):
      terms_comp = zip(terms, comp)
      sorted_terms = sorted(terms_comp, key= lambda x:x[1],
reverse=True)[:7]
      print("Topic "+str(i)+": ")
      for t in sorted_terms:
            print(t[0])
      print(" ")
```

Questions:

1. Try changing the input data to observe generation of different topics based on similarity.
2. Vary the number of topics generated (n_components) to compare and contrast them. Are there any similarity between topics? What happens as the number of topics increase/decrease?
3. Modify parameters of 'vectorizer' such as max_df and observe how the output is affected.

Additional Reading Material

1. Latent Semantic Analysis
2. An example for LSA (R code for LSA)
3. Sentiment Analysis using LSA - Good code with explanation
4. LSA documentation in R
5. Vanilla PCA Refresher
6. Sparse Principal Component Analysis Paper
7. Why SPCA over PCA
8. SPCA Python Documentation - Read to learn about the various parameters and how they affect SPCA
9. Package 'sparsepca' (R code for SPCA)
10. Why do we center the data when doing PCA?

A/B Testing and Interpreting Business Values

I. Confounding Variables and Simpson's Paradox

A discrimination lawsuit was filed against the California DDS, claiming that White Non-Hispanics were receiving more funding than Hispanics. Using the given dataset and the concept of Simpson's Paradox, determine if there is Ethnic bias in the Expenditures. This data set contains data regarding the allocation of funding from the Department of Developmental Services to developmentally-disabled individuals in California in 2014.

Data:- California DDS Expenditures

Code:-

```
#Step 1: Load the necessary packages:

import pandas as pd
import seaborn as sns

#Step 2: Load the data set:
path = /path/to/data/
test_df=pd.read_csv(path+"california.csv")

# Step 3: Our problem statement is to determine if there is Ethnic bias
in the Expenditures.
# Let us find the mean of Expenditures per Ethnicity and check if the
claim is valid.

a = test_df.groupby('Ethnicity').mean().sort_values('Expenditures')
print(a)

# This will show that there is some variation in expenditures according
to ethnicity, so we will plot a boxplot to visualise the same

# Step 4: A box plot for the above data is shown below.
bplot=sns.boxplot(y='Ethnicity', x='Expenditures',
                  data=test_df.sort_values('Ethnicity'),
                  width=0.5,
                  palette="colorblind")
bplot.axes.set_title("Expenditures By Ethnicity", fontsize=16)
bplot.set_xlabel("Expenditures", fontsize=14)
bplot.set_ylabel("Ethnicity", fontsize=14)
```

```
bplot.tick_params(labelsize=10)

# From the above graph, it looks like there is discrimination towards
multiple ethnic groups.
# Let us divide the data based on other features to check if the above
result is accurate

#Step 5: Let us begin with grouping the Expenditures data on Gender:
test_df.loc[:,['Gender',
'Expenditures']].groupby('Gender').mean().sort_values('Gender')

# This will show that there is no significant difference in Expenditure
with respect to Male and Female. This means we can ignore the Gender
feature in our future analysis.

#Step 6: Let us consider Age. We will use the Age Cohort feature
available in the data set.
test_df.loc[:,['Age Cohort', 'Expenditures']].groupby('Age
Cohort').mean().sort_values('Expenditures')


# The above result will show that the fund allocations were performed
on the basis of Age.

df = test_df.loc[:,['Age Cohort', 'Expenditures', 'Ethnicity',
'Expenditures_categorical']]

#Step 7: Let us get a perspective of how funds are allocated to
different Ethnicities within the Age groups.

zero_to_5 = df['Age Cohort'] == '0 to 5'
six_to_12 = df['Age Cohort'] == '6 to 12'
thirteen_to_17 = df['Age Cohort'] == '13 to 17'
eighteen_to_21 = df['Age Cohort'] == '18 to 21'
twentytwo_to_50 = df['Age Cohort'] == '22 to 50'
fiftyone_plus = df['Age Cohort'] == '51+'

#Step 8: Graph a box plot for the data based on age group. Draw your
conclusions by comparing these graphs to the one plotted above.

bplot=sns.boxplot(y='Ethnicity', x='Expenditures',
data=df.where(zero_to_5).dropna().sort_values('Ethnicity'),
width=0.5,
```

```
                palette="colorblind")
bplot.axes.set_title("Expenditures By Ethnicity, Age: 0 to 5",
fontsize=16)
bplot.set_xlabel("Expenditures", fontsize=14)
bplot.set_ylabel("Ethnicity", fontsize=14)
bplot.tick_params(labelsize=10)
bplot=sns.boxplot(y='Ethnicity', x='Expenditures',
            data=df.where(six_to_12).dropna().sort_values('Ethnicity'),
                width=0.5,
                palette="colorblind")
bplot.axes.set_title("Expenditures By Ethnicity, Age: 6 to 12",
fontsize=16)
bplot.set_xlabel("Expenditures", fontsize=14)
bplot.set_ylabel("Ethnicity", fontsize=14)
bplot.tick_params(labelsize=10)
```

Questions:

1. Plot the graph for all age groups to compare the distribution of funds for different age groups. The plots for two groups have been drawn for you.
2. What conclusions can you draw based on the above box plots? Are the claims made by the lawsuit valid?
3. What is the confounding variable in this dataset?

II. A/B Testing

You are a data analyst in an E-Commerce company. Your company wants to add a feature to their website and they ask you to find out if the feature will improve the conversion rate to 12%, from 10%. You decide to use A/B Testing to find out. After releasing the control version to group A and the new version to group B, you get the following data :

ABdata

Now you need to find out if the feature really helped improve conversion rate.

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats
```

```
path = 'path/to/data/'

data = pd.read_csv(path+'abdata.csv')

# Number of Users who tried A vs B
count = data.groupby(["group"]).count()
n_a = count["converted"][0]
n_b = count["converted"][1]
# Number of users that converted into a purchase for A vs B
converted = data.groupby(["group"]).sum()

# Proportion of group A conversions
p_a = converted["converted"][0]/n_a
# Proportion of group B conversions
p_b = converted["converted"][1]/n_b

"""Our null hypothesis is that p_a=p_b, which is to say they come
from the same distribution. In order to show that the feature
works we need to disprove the null hypothesis"""

# Standard deviation of A and B
sigma_a = np.sqrt((p_a*(1-p_a))/n_a)
sigma_b = np.sqrt((p_b*(1-p_b))/n_b)

# Plotting our distributions
x = np.linspace(p_a - 3*sigma_a, p_b + 3*sigma_b,100)
y1 =  stats.norm.pdf(x, p_a, sigma_a)
y2 =  stats.norm.pdf(x, p_b, sigma_b)
plt.plot(x, y1,'r')
plt.plot(x, y2,'b')
plt.show()

# Now we calculate pooled probability (to calculate sigma of pa-
pb) - read about Sattherwaithe approximation
# Don't get scared by the fancy name - it's a simple concept :)

pooled_prob = (p_a*n_a + p_b*n_b)/(n_a+ n_b)
reciprocal_count_sum = (1/n_a + 1/n_b)

in_root = (pooled_prob*(1-pooled_prob))*reciprocal_count_sum

pooled_sigma = np.sqrt(in_root)

# Take p_threshold as 0.05

# Calculating p_value
z_score = (p_b - p_a)/(pooled_sigma)
p_value = stats.norm.sf(z_score)
```

Questions:

1. Write additional code to find the power of this Test (remember Power of a Test is the probability that we will correctly reject null hypothesis, i.e 1 - Type II error)
2. Is the power sufficiently high? How do we increase the power of the test?
3. All things considered, would you recommend implementing the new feature?

Additional Reading Material

1. Confounding variables
2. How to visualize hidden relationships in data with Python — analysing NBA assists
3. AB testing python code
4. Evaluating the Business Value of Predictive Models in Python and R
5. A/B TESTING FOR MARKETING OPTIMIZATION
6. A/B Testing (Hypothesis Testing)
7. Using Analytics To Identify The Business Value of Your Website
8. Some Good KPIs to track
9. Some More A/B testing python code
10. Finding the Power of a Hypothesis Test
11. The Importance of A/B Testing
12. When A/B Testing Isn't Worth It
13. Statistical Power and Power Analysis (Uses inbuilt python functions)
14. Hypothesis Testing with two independent samples