Assignment 2 for Course 1MS041

Make sure you pass the `# ... Test` cells and submit your solution notebook in the corresponding assignment on the course website. You can submit multiple times before the deadline and your highest score will be used.

Assignment 2, PROBLEM 1

Maximum Points = 8

A courier company operates a fleet of delivery trucks that make deliveries to different parts of the city. The trucks are equipped with GPS tracking devices that record the location of each truck at regular intervals. The locations are divided into three regions: downtown, the suburbs, and the countryside. The following table shows the probabilities of a truck transitioning between these regions at each time step:

Current region	Probability of transitioning to downtown	Probability of transitioning to the suburbs	Probability of transitioning to the countryside
Downtown	0.3	0.4	0.3
Suburbs	0.2	0.5	0.3
Countryside	0.4	0.3	0.3

1. If a truck is currently in the suburbs, what is the probability that it will be in the downtown region after two time steps? [1.5p]
2. If a truck is currently in the suburbs, what is the probability that it will be in the downtown region **the first time** after two time steps? [1.5p]
3. Is this Markov chain irreducible? [1.5p]
4. What is the stationary distribution? [1.5p]
5. Advanced question: What is the expected number of steps until the first time one enters the downtown region having started in the suburbs region. Hint: to get within 1 decimal point, it is enough to compute the probabilities for hitting times below 30. [2p]

In [ ]:
```
# Part 1

# Fill in the answer to part 1 below as a decimal number
problem1_p1 = XXX
```

In [ ]:
```
# Part 2

# Fill in the answer to part 2 below as a decimal number
problem1_p2 = XXX
```

In [ ]:
```
# Part 3

# Fill in the answer to part 3 below as a boolean
problem1_irreducible = True/False
```

In [ ]:
```
# Part 4

# Fill in the answer to part 4 below
# the answer should be a numpy array of length 3
# make sure that the entries sums to 1!
problem1_stationary = XXX
```

In [ ]:
```
# Part 5

# Fill in the answer to part 5 below
# That is, the expected number of steps as a decimal number
problem1_ET = XXX
```

Assignment 2, PROBLEM 2

Maximum Points = 8

A healthcare organization is interested in understanding the relationship between the number of visits to the doctors office and certain patient characteristics. They have collected data on the number of visits (for a year) for a sample of patients and have included the following variables

- ofp : number of physician office visits
- ofnp : number of nonphysician office visits
- opp : number of physician outpatient visits
- opnp : number of nonphysician outpatient visits
- emr : number of emergency room visits
- hosp : number of hospitalizations
- exclhlth : the person is of excellent health (self-perceived)
- poorhealth : the person is of poor health (self-perceived)
- numchron : number of chronic conditions
- adldiff : the person has a condition that limits activities of daily living ?
- noreast : the person is from the north east region
- midwest : the person is from the midwest region

- west : the person is from the west region
- age : age in years (divided by 10)
- male : is the person male ?
- married : is the person married ?
- school : number of years of education
- faminc : family income in 10000$
- employed : is the person employed ?
- privins : is the person covered by private health insurance?
- medicaid : is the person covered by medicaid ?

Decide which patient features are resonable to use to predict the target "number of physician office visits". Hint: should we really use the "ofnp" etc variables?

Since the target variable is counts, a reasonable loss function is to consider the target variable as Poisson distributed where the parameter follows $\lambda = \exp(\alpha \cdot x + \beta)$ where α is a vector (slope) and β is a number (intercept). That is, the parameter is the exponential of a linear function. The reason we chose this as our parameter, is that it is always positive which is when the Poisson distribution is defined. To be specific we make the following assumption about our conditional density of $Y \mid X$,

$$f_{Y|X}(y, x) = \frac{\lambda^y e^{-\lambda}}{y!}, \quad \lambda(x) = \exp(\alpha \cdot x + \beta).$$

Recall from the lecture notes, (4.2) that in this case we should consider the log-loss (entropy) and that according to (4.2.1 Maximum Likelihood and regression) we can consider the conditional log-likelihood. Follow the steps of Example 1 and Example 2 in section (4.2) to derive the loss that needs to be minimized.

Hint: when taking the log of the conditional density you will find that the term that contains the $y!$ does not depend on λ and as such does not depend on α, β, it can thus be discarded. This will be essential due to numerical issues with factorials.

Instructions:

1. Load the file `data/visits_clean.csv` , follow the instructions in the code cell of how this should happen [1.5p]
2. Create the `problem2_X` and the `problem2_y` as numpy arrays with `problem2_X` being the features and `problem2_y` being the target. Do the standard train-test split with 80% training data and 20% testing data. Store these in the variables defined in the cells. [1.5p]
3. Implement $loss$ inside the class `PoissonRegression` by writing down the loss to be minimized, I have provided a formula for the λ that you can use. [1.5p]
4. Now use the `PoissonRegression` class to train a Poisson regression model on the training data. [1.5p]
5. Compute the mean absolute error of your prediction on the test set and use Hoeffdings inequality to produce a 95\% confidence interval for the mean absolute error. We can make the assumption that the error is bounded by 70 for simplicity. [2p]

In [ ]:
```
# Part 1

# As in assignment 1 we will load the header into header and data i
nto data
# this time you will have to parse the data such that each data ent
ry is a float
# and that the problem2_data is a numpy array of shape (n_samples,n
_columns)
# where n_columns is the number of columns and should have the same
length as
# the list of strings header. n_samples is how many rows of data we
had.
# If you cannot find the file, check the starting package as it sho
uld be updated
# if not, go to the github repo and pull it

# The autograder does not accept pandas as a solution to this probl
em.
# data/visits_clean.csv
problem2_header = [""] #List of strings
problem2_data = np.array() #A numpy array of shape n_samples n_colu
mns
```

In [ ]:
```
# Part 2

# Fill in your X and y below
problem2_X = XXX
problem2_y = XXX

# Split the data into train and randomly using for instance
# np.random.shuffle indices and indexing the first 80% as the train
data
# keep the train size as 0.8 rounded up to the nearest integer samp
le
problem2_X_train, problem2_X_test, problem2_y_train, problem2_y_tes
t = XXX
```

In [ ]:
```
# Part 3

# Fill in the function loss below

class PoissonRegression(object):
    def __init__(self):
        self.coeffs = None
        self.result = None

    # define the objective/cost/loss function we want to minimise
    def loss(self,X,Y,coeffs):
        # The parameter lambda for the given X and the proposed val
ues
        # of the coefficients, here coeff[:-1] represent alpha
        # and coeff[-1] represent beta
        lam = XXX

        # use the Y variable that is available here to define
        # the loss function, return the value of the loss for
        # this Y and for this parameter lam defined above
        return XXX

    def fit(self,X,Y):
        import numpy as np
        from scipy import optimize

        #Use the loss above together with an optimization method fr
om scipy
        #to find the coefficients of the model
        #this is prepared for you below
        opt_loss = lambda coeffs: self.loss(X,Y,coeffs)
        initial_arguments = np.zeros(shape=X.shape[1]+1) # initial
guess as 0
        self.result = optimize.minimize(opt_loss, initial_argument
s,method='cg')
        self.coeffs = self.result.x

    def predict(self,X):
        #Use the trained model to predict Y
        if (self.coeffs is not None):
            return np.exp(np.dot(X,self.coeffs[:-1])+self.coeffs[-
1])
```

In [ ]:
```
# Part 4

# Initialize your PoissonRegression model
problem2_model = XXX

# Fit your initialized model on the training data


# This is to make sure that everything went well,
# check that success is True
print(problem2_model.result)
```

In [ ]:
```
# Part 5

# Put the computed mean absolute error in the variable below
problem2_metric = MAE
# Put a confidence interval in the variable below by using Hoeffdin
gs inequality using the bounds
# a = 0, b=70 (roughly 5 days between visits as minimum)
# the variable should contain a tuple representing the confidence i
nterval of the form (l_edge,r_edge)
problem2_interval = XXX
```

Assignment 2, PROBLEM 3

Maximum Points = 8

Random variable generation and transformation

The purpose of this problem is to show that you can implement your own sampler, this will be built in the following three steps:

1. [2p] Implement a Linear Congruential Generator where you tested out a good combination (a large M with a, b satisfying the Hull-Dobell (Thm 6.8)) of parameters. Follow the instructions in the code block.
2. [2p] Using a generator construct random numbers from the uniform $[0, 1]$ distribution.
3. [4p] Using a uniform $[0, 1]$ random generator, generate samples from

$$p_0(x) = \frac{\pi}{2}|\sin(2\pi x)|, \quad x \in [0, 1] \ .$$

Using the **Accept-Reject** sampler (**Algorithm 1** in TFDS notes) with sampling density given by the uniform $[0, 1]$ distribution.

In [ ]:
```
def problem3_LCG(size=None, seed = 0):
    """
    A linear congruential generator that generates pseudo random nu
mbers according to size.

    Parameters
    -------------
    size : an integer denoting how many samples should be produced
    seed : the starting point of the LCG, i.e. u0 in the notes.

    Returns
    -------------
    out : a list of the pseudo random numbers
    """

    XXX

    return XXX
```

In [ ]:
```
def problem3_uniform(generator=None, period = 1, size=None, seed=
0):
    """
    Takes a generator and produces samples from the uniform [0,1] d
istribution according
    to size.

    Parameters
    -------------
    generator : a function of type generator(size,seed) and produce
s the same result as problem1_LCG, i.e. pseudo random numbers in th
e range {0,1,...,period-1}
    period : the period of the generator
    seed : the seed to be used in the generator provided
    size : an integer denoting how many samples should be produced

    Returns
    --------------
    out : a list of the uniform pseudo random numbers
    """

    XXX

    return XXX
```

In [ ]:
```
def problem3_accept_reject(uniformGenerator=None, n_iterations=None, seed=0):
    """
    Takes a generator that produces uniform pseudo random [0,1] numbers
    and produces samples from (pi/2)*abs(sin(x*2*pi)) using an Accept-Reject
    sampler with the uniform distribution as the proposal distribution.
    Runs n_iterations

    Parameters
    -------------
    generator : a function of the type generator(size,seed) that produces uniform pseudo random
    numbers from [0,1]
    seed : the seed to be used in the generator provided
    n_iterations : an integer denoting how many attempts should be made in the accept-reject sampler

    Returns
    -------------
    out : a list of the pseudo random numbers with the specified distribution
    """

    XXX

    return XXX
```

Local Test for Assignment 2, PROBLEM 3

Evaluate cell below to make sure your answer is valid. You **should not** modify anything in the cell below when evaluating it to do a local test of your solution. You may need to include and evaluate code snippets from lecture notebooks in cells above to make the local test work correctly sometimes (see error messages for clues). This is meant to help you become efficient at recalling materials covered in lectures that relate to this problem. Such local tests will generally not be available in the exam.

In [ ]:
```
# If you managed to solve all three parts you can test the followin
g code to see if it runs
# you have to change the period to match your LCG though, this is m
arked as XXX.
# It is a very good idea to check these things using the histogram
function in sagemath
# try with a larger number of samples, up to 10000 should run

print("LCG output: %s" % problem3_LCG(size=10, seed = 1))

period = XXX

print("Uniform sampler %s" % problem3_uniform(generator=problem3_LC
G, period = period, size=10, seed=1))

uniform_sampler = lambda size,seed: problem3_uniform(generator=prob
lem3_LCG, period = period, size=size, seed=seed)

print("Accept-Reject sampler %s" % problem3_accept_reject(uniformGe
nerator = uniform_sampler,n_iterations=20,seed=1))
```

In [ ]:
```
# If however you did not manage to implement either part 1 or part
2 but still want to check part 3, you can run the code below

def testUniformGenerator(size,seed):
    import random
    random.seed(seed)

    return [random.uniform(0,1) for s in range(size)]

print("Accept-Reject sampler %s" % problem3_accept_reject(uniformGe
nerator=testUniformGenerator, n_iterations=20, seed=1))
```