Assignment 4 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.

keyboard\_arrow\_down

Assignment 4, PROBLEM 1

Maximum Points = 24

This time the assignment only consists of one problem, but we will do a more comprehensive analysis instead.

Consider the dataset Corona\_NLP\_train.csv that you can get from the course website [git](https://github.com/datascience-intro/1MS041-2024/blob/main/notebooks/data/Corona_NLP_train.csv" \t "_blank). The data is "Coronavirus tweets NLP - Text Classification" that can be found on [kaggle](https://www.google.com/url?q=https%3A%2F%2Fwww.kaggle.com%2Fdatasets%2Fdatatattle%2Fcovid-19-nlp-text-classification" \t "_blank). The data has several columns, but we will only be working with OriginalTweetand Sentiment.

1. [3p] Load the data and filter out those tweets that have Sentiment=Neutral. Let X represent the OriginalTweet and let $$

 Y =  
 \begin{cases}

1 & \text{if sentiment is towards positive}

\\

0 & \text{if sentiment is towards negative}.

\end{cases}

$$ Put the resulting arrays into the variables X and Y. Split the data into three parts, train/test/validation where train is 60% of the data, test is 15% and validation is 25% of the data. Do not do this randomly, this is to make sure that we all did the same splits (we are in this case assuming the data is IID as presented in the dataset). That is [train,test,validation] is the splitting layout.

1. [4p] There are many ways to solve this classification problem. The first main issue to resolve is to convert the X variable to something that you can feed into a machine learning model. For instance, you can first use [CountVectorizer](https://www.google.com/url?q=https%3A%2F%2Fscikit-learn.org%2F1.5%2Fmodules%2Fgenerated%2Fsklearn.feature_extraction.text.CountVectorizer.html" \t "_blank) as the first step. The step that comes after should be a LogisticRegression model, but for this to work you need to put together the CountVectorizer and the LogisticRegression model into a [Pipeline](https://www.google.com/url?q=https%3A%2F%2Fscikit-learn.org%2F1.5%2Fmodules%2Fgenerated%2Fsklearn.pipeline.Pipeline.html%23sklearn.pipeline.Pipeline). Fill in the variable model such that it accepts the raw text as input and outputs a number 0 or 1, make sure that model.predict\_proba works for this. **Hint: You might need to play with the parameters of LogisticRegression to get convergence, make sure that it doesn't take too long or the autograder might kill your code**
2. [3p] Use your trained model and calculate the precision and recall on both classes. Fill in the corresponding variables with the answer.
3. [3p] Let us now define a cost function
   * A positive tweet that is classified as negative will have a cost of 1
   * A negative tweet that is classified as positive will have a cost of 5
   * Correct classifications cost 0

complete filling the function cost to compute the cost of a prediction model under a certain prediction threshold (recall our precision recall lecture and the predict\_proba function from trained models).

1. [4p] Now, we wish to select the threshold of our classifier that minimizes the cost, fill in the selected threshold value in value optimal\_threshold.
2. [4p] With your newly computed threshold value, compute the cost of putting this model in production by computing the cost using the validation data. Also provide a confidence interval of the cost using Hoeffdings inequality with a 99% confidence.
3. [3p] Let t be the threshold you found and f the model you fitted (one of the outputs of predict\_proba), if we define the random variable $$

 C = (1-1\_{f(X)\geq t})Y+5(1-Y)1\_{f(X) \geq t}

$$ then C denotes the cost of a randomly chosen tweet. In the previous step we estimated E[C] using the empirical mean. However, since the threshold is chosen to minimize cost it is likely that C=0 or C=1 than C=5 as such it will have a low variance. Compute the empirical variance of C on the validation set. What would be the confidence interval if we used Bennett's inequality instead of Hoeffding in point 6 but with the computed empirical variance as our guess for the variance?

[ ]

# Part 1  
  
# Load the data from the file specified in the problem definition and make sure that it is loaded using  
# the search path `data/Corona\_NLP\_train.csv`. This is to make sure the autograder and your computer have the same  
# file path and can load the data correctly.  
  
# Contrary to how many other problems are structured, this problem actually requires you to  
# have X on the shape (n\_samples, ) that is a 1-dimensional array. Otherwise it will cause a bunch  
# of errors in the autograder or also in for instance CountVectorizer.  
  
# Make sure that all your data is numpy arrays and not pandas dataframes or series.  
  
import numpy as np  
import pandas as pd  
import math  
  
# Import CSV file  
corona\_raw = pd.read\_csv('data/Corona\_NLP\_train.csv', delimiter=',', encoding='latin1')  
corona=corona\_raw[corona\_raw['Sentiment'] != 'Neutral']  
X=corona['OriginalTweet'].to\_list()  
sentiment\_map = {  
    "Positive": 1,  
    "Extremely Positive": 1,  
    "Negative": 0,  
    "Extremely Negative": 0  
}  
Y= corona['Sentiment'].map(sentiment\_map)  
# Calculate the indices for splitting  
train\_size = int(0.60 \* len(corona))  
test\_size = int(0.15 \* len(corona))  
  
# Split the data sequentially  
X\_train = X[:train\_size]  
X\_test = X[train\_size:train\_size + test\_size]  
X\_valid = X[train\_size + test\_size:]  
  
Y\_train = Y[:train\_size]  
Y\_test = Y[train\_size:train\_size + test\_size]  
Y\_valid= Y[train\_size + test\_size:]

[ ]

# Part 2  
  
# Train a machine learning model or pipeline that can take the raw strings from X and predict Y=0,1 depending on the  
# sentiment of the tweet. Store the trained model in the variable `model`.  
  
from sklearn.feature\_extraction.text import CountVectorizer  
from sklearn.linear\_model import LogisticRegression  
from sklearn.pipeline import Pipeline  
  
  
pipeline = Pipeline([  
    ('vectorizer', CountVectorizer()),  # Convert text to feature matrix  
    ('classifier', LogisticRegression(solver='saga', C=0.1))  # Train logistic regression  
                   ])  
  
model = pipeline.fit(X,Y)

[ ]

from sklearn.metrics import precision\_score, recall\_score  
  
Y\_pred = model.predict(X\_test)  
precision\_0 = precision\_score(Y\_test, Y\_pred, pos\_label=0)  
precision\_1 = precision\_score(Y\_test, Y\_pred, pos\_label=1)  
recall\_0 = recall\_score(Y\_test, Y\_pred, pos\_label=0)  
recall\_1 = recall\_score(Y\_test, Y\_pred, pos\_label=1)

[ ]

# Part 4  
  
def cost(model,threshold,X,Y):  
    # Hint, make sure that the model has a predict\_proba method  
    # think about how the decision is made based on the probabilities  
    # and how the threshold can be used to make the decision.  
    # For reference take a look at the lecture notes "Bayes classifier"  
    # which contains how the decision is made based on the probabilities when the threshold is 0.5.  
  
    # Fill in what is missing to compute the cost and return it  
    # Note that we are interested in average cost  
    probs = model.predict\_proba(X)[:, 1]  
    predictions = (probs >= threshold).astype(int)  
    false\_negatives = (Y == 1) & (predictions == 0)  
    false\_positives = (Y == 0) & (predictions == 1)  
    total\_cost = np.sum(false\_negatives) \* 1 + np.sum(false\_positives) \* 5  
    avg\_cost = total\_cost / len(Y)  
  
  
  
  
    return avg\_cost

[ ]

# Part 5  
  
# Find the optimal threshold for the model on the test set. Store the threshold in the variable `optimal\_threshold`  
# and the cost at the optimal threshold in the variable `cost\_at\_optimal\_threshold` evaluated on the test set.  
import numpy as np  
thresholds = np.linspace(0.0, 1.0, 101)  
costs = [cost(model, threshold, X\_test, Y\_test) for threshold in thresholds]  
  
optimal\_threshold = thresholds[np.argmin(costs)]  
cost\_at\_optimal\_threshold = costs[np.argmin(costs)]

[ ]

# Part 6  
cost\_validation=cost(model, optimal\_threshold, X\_valid, Y\_valid)  
n = len(Y\_valid)  # Number of validation samples  
epsilon = 0.01  
#Hoeffding parameters  
a = 0  
b = 5  
confidence\_level = 0.99  
delta = 1 - confidence\_level  
#HOEFFDING'S BOUND  
hoeff\_bound= math.sqrt((math.log(2 / delta) \* (b - a) \*\* 2) / (2 \* n))  
lower\_bound=cost\_validation-hoeff\_bound  
upper\_bound=cost\_validation+hoeff\_bound  
cost\_at\_optimal\_threshold\_valid = cost(model, optimal\_threshold, X\_valid, Y\_valid)  
cost\_interval\_valid = (lower\_bound, upper\_bound)  
  
assert(type(cost\_interval\_valid) == tuple)  
assert(len(cost\_interval\_valid) == 2)

arrow\_upwardarrow\_downward

spark

link

comment

settings

delete

more\_vert

[ ]

# Part 7

import numpy as np

import math

probs\_valid = model.predict\_proba(X\_valid)[:, 1]  # positive class

predictions\_valid = (probs\_valid>= optimal\_threshold).astype(int)

C\_valid = (1 - predictions\_valid) \* Y\_valid + 5 \* (1 - Y\_valid) \* predictions\_valid

variance\_of\_C =  np.var(C\_valid)

mean\_C = np.mean(C\_valid)

n = len(C\_valid)

confidence\_level = 0.99

delta = 1 - confidence\_level

# Bennett's inequality bound

epsilon\_bound = math.sqrt((2 \* variance\_of\_C \* math.log(1 / delta)) / n)

interval\_of\_C = (mean\_C - epsilon\_bound, mean\_C + epsilon\_bound)

assert(type(interval\_of\_C) == tuple)

assert(len(interval\_of\_C) == 2)

# Assignment 3 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.

keyboard\_arrow\_down

## Assignment 3, PROBLEM 1

Maximum Points = 8

Download the updated data folder from the course github website or just download directly the file <https://github.com/datascience-intro/1MS041-2024/blob/main/notebooks/data/smhi.csv> from the github website and put it inside your data folder, i.e. you want the path data/smhi.csv. The data was aquired from SMHI (Swedish Meteorological and Hydrological Institute) and constitutes per hour measurements of wind in the Uppsala Aut station. The data consists of windspeed and direction. Your goal is to load the data and work with it a bit. The code you produce should load the file as it is, please do not alter the file as the autograder will only have access to the original file.

The file information is in Swedish so you need to use some translation service, for instance Google translate or ChatGPT.

1. [2p] Load the file, for instance using the csv package. Put the wind-direction as a numpy array and the wind-speed as another numpy array.
2. [2p] Use the wind-direction which is an angle in degrees and convert it into a point on the unit circle. Store the x\_coordinate as one array and the y\_coordinate as another. From these coordinates, construct the wind-velocity vector.
3. [2p] Calculate the average wind velocity and convert it back to direction and compare it to just taking average of the wind direction as given in the data-file.
4. [2p] The wind velocity is a 2-dimensional random variable, calculate the empirical covariance matrix which should be a numpy array of shape (2,2).

For you to wonder about, is it more likely for you to have headwind or not when going to the university in the morning.

[ ]

import numpy as np  
import pandas as pd  
import math  
  
# Import CSV file  
smhi = pd.read\_csv('data/smhi.csv', skiprows=11,delimiter=';')  
  
problem1\_wind\_direction = smhi['Vindriktning'].to\_numpy()  
problem1\_wind\_speed = smhi['Vindhastighet'].to\_numpy()

[ ]

# The wind direction is given as a compass direction in degrees (0-360)  
# convert it to x and y coordinates using the standard mathematical convention  
problem1\_wind\_direction\_x\_coordinate = np.cos(np.radians(problem1\_wind\_direction))  
problem1\_wind\_direction\_y\_coordinate = np.sin(np.radians(problem1\_wind\_direction))  
  
  
problem1\_wind\_velocity\_x\_coordinate = problem1\_wind\_direction\_x\_coordinate \* problem1\_wind\_speed  
problem1\_wind\_velocity\_y\_coordinate = problem1\_wind\_direction\_y\_coordinate \* problem1\_wind\_speed

[ ]

# Put the average wind velocity x and y coordinates here in these variables  
problem1\_average\_wind\_velocity\_x\_coordinate = np.mean(problem1\_wind\_velocity\_x\_coordinate)  
problem1\_average\_wind\_velocity\_y\_coordinate = np.mean(problem1\_wind\_velocity\_y\_coordinate)  
  
# First calculate the angle of the average wind velocity vector in degrees  
problem1\_average\_wind\_velocity\_angle\_degrees =np.degrees(np.arctan2(problem1\_average\_wind\_velocity\_y\_coordinate, problem1\_average\_wind\_velocity\_x\_coordinate))  
# Then calculate the average angle of the wind direction in degrees (using the wind direction in the data)  
problem1\_average\_wind\_direction\_angle\_degrees = np.mean(problem1\_wind\_direction)  
  
# Finally, are they the same? Answer as a boolean value (True or False)  
problem1\_same\_angle = False

[ ]

problem1\_wind\_velocity\_covariance\_matrix =  np.cov(np.vstack((problem1\_wind\_velocity\_x\_coordinate, problem1\_wind\_velocity\_y\_coordinate)))

keyboard\_arrow\_down

## Assignment 3, PROBLEM 2

Maximum Points = 8

For this problem you will need the [pandas](https://www.google.com/url?q=https%3A%2F%2Fpandas.pydata.org%2F) package and the [sklearn](https://www.google.com/url?q=https%3A%2F%2Fscikit-learn.org%2Fstable%2F" \t "_blank) package. Inside the data folder from the course website you will find a file called indoor\_train.csv, this file includes a bunch of positions in (X,Y,Z) and also a location number. The idea is to assign a room number (Location) to the coordinates (X,Y,Z).

1. [2p] Take the data in the file indoor\_train.csv and load it using pandas into a dataframe df\_train
2. [3p] From this dataframe df\_train, create two numpy arrays, one Xtrain and Ytrain, they should have sizes (1154,3) and (1154,) respectively. Their dtype should be float64 and int64 respectively.
3. [3p] Train a Support Vector Classifier, sklearn.svc.SVC, on Xtrain, Ytrain with kernel='linear' and name the trained model svc\_train.

To mimic how [kaggle](https://www.google.com/url?q=https%3A%2F%2Fwww.kaggle.com%2F" \t "_blank) works, the Autograder has access to a hidden test-set and will test your fitted model.

[ ]

import pandas as pd  
  
df\_train = pd.read\_csv('data/indoor\_train.csv',delimiter=',')

[ ]

Xtrain = df\_train.iloc[:,:3].to\_numpy()  
Ytrain = df\_train.iloc[:,3].to\_numpy()

[ ]

from sklearn.svm import SVC  
  
svc\_train = SVC(kernel='linear')  
svc\_train.fit(Xtrain,Ytrain)

keyboard\_arrow\_down

## Assignment 3, PROBLEM 3

Maximum Points = 8

Let us build a proportional model (P(Y=1∣X)=G(β0+β⋅X) where G is the logistic function) for the spam vs not spam data. Here we assume that the features are presence vs not presence of a word, let X1,X2,X3 denote the presence (1) or absence (0) of the words ("free","prize","win").

1. [2p] Load the file data/spam.csv and create two numpy arrays, problem3\_X which has shape **(n\_texts,3)** where each feature in problem3\_X corresponds to X1,X2,X3 from above, problem3\_Y which has shape **(n\_texts,)** and consists of a 1 if the email is spam and 0 if it is not. Split this data into a train-calibration-test sets where we have the split 40%, 20%, 40%, put this data in the designated variables in the code cell.
2. [2p] Follow the calculation from the lecture notes where we derive the logistic regression and implement the final loss function inside the class ProportionalSpam. You can use the Test cell to check that it gives the correct value for a test-point.
3. [2p] Train the model problem3\_ps on the training data. The goal is to calibrate the probabilities output from the model. Start by creating a new variable problem3\_X\_pred (shape (n\_samples,1)) which consists of the predictions of problem3\_ps on the calibration dataset. Then train a calibration model using sklearn.tree.DecisionTreeRegressor, store this trained model in problem3\_calibrator. Recall that calibration error is the following for a fixed function f

E[|E[Y∣f(X)]−f(X)|2]−−−−−−−−−−−−−−−−−−−√.

1. [2p] Use the trained model problem3\_ps and the calibrator problem3\_calibrator to make final predictions on the testing data, store the prediction in problem3\_final\_predictions.

[ ]

import numpy as np  
import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
  
# Load the data  
data = pd.read\_csv('data/spam.csv', delimiter=',',encoding='latin1')  
v1=data['v1'].to\_numpy()  
free=data['v2'].str.contains(r'\bfree\b', case=False, na=False).to\_numpy()  
prize=data['v2'].str.contains(r'\bprize\b', case=False, na=False).to\_numpy()  
win=data['v2'].str.contains(r'\bwin\b', case=False, na=False).to\_numpy()  
  
  
problem3\_X = np.vstack([free,prize,win]).T  
problem3\_Y = v1=='spam'  
  
#trai\_calib\_slpit  
problem3\_train\_cal, problem3\_X\_test,problem3\_Y\_train\_cal , problem3\_Y\_test=train\_test\_split(  
    problem3\_X, problem3\_Y, test\_size=0.4, random\_state=42)  
problem3\_X\_train,problem3\_X\_calib,problem3\_Y\_train,problem3\_Y\_calib=train\_test\_split( problem3\_train\_cal,problem3\_Y\_train\_cal,test\_size=0.33, random\_state=42)  
#assign  
  
  
print(problem3\_X\_train.shape,problem3\_X\_calib.shape,problem3\_X\_test.shape,problem3\_Y\_train.shape,problem3\_Y\_calib.shape,problem3\_Y\_test.shape)

(2239, 3) (1104, 3) (2229, 3) (2239,) (1104,) (2229,)

[ ]

class ProportionalSpam(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):  
        bias = coeffs[0]  
        weights = coeffs[1:]  
        z = np.dot(X, weights) + bias  
        p = np.exp(z) / (1 + np.exp(z))  
        log\_loss = -np.mean(Y \* np.log(p) + (1 - Y) \* np.log(1 - p))  
  
        return log\_loss  
  
    def fit(self,X,Y):  
        import numpy as np  
        from scipy import optimize  
  
        #Use the f above together with an optimization method from scipy  
        #to find the coefficients of the model  
        opt\_loss = lambda coeffs: self.loss(X,Y,coeffs)  
        initial\_arguments = np.zeros(shape=X.shape[1]+1)  
        self.result = optimize.minimize(opt\_loss, initial\_arguments,method='cg')  
        self.coeffs = self.result.x  
  
    def predict(self,X):  
        #Use the trained model to predict Y  
        if (self.coeffs is not None):  
            G = lambda x: np.exp(x)/(1+np.exp(x))  
            return np.round(10\*G(np.dot(X,self.coeffs[1:])+self.coeffs[0]))/10 # This rounding is to help you with the calibration

[ ]

problem3\_ps = ProportionalSpam()  
problem3\_ps.fit(problem3\_X\_train, problem3\_Y\_train)  
  
  
  
problem3\_X\_pred = problem3\_ps.predict(problem3\_X\_calib).reshape(-1, 1)  
from sklearn.tree import DecisionTreeRegressor  
  
calib=DecisionTreeRegressor()  
problem3\_calibrator = calib.fit(problem3\_X\_pred,problem3\_Y\_calib)

[ ]

test\_predictions\_proba = problem3\_ps.predict(problem3\_X\_test).reshape(-1, 1)  
problem3\_final\_predictions = problem3\_calibrator.predict(test\_predictions\_proba)

keyboard\_arrow\_down

#### Local Test for Assignment 3, 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.

[ ]

try:  
    import numpy as np  
    test\_instance = ProportionalSpam()  
    test\_loss = test\_instance.loss(np.array([[1,0,1],[0,1,1]]),np.array([1,0]),np.array([1.2,0.4,0.3,0.9]))  
    assert (np.abs(test\_loss-1.2828629432232497) < 1e-6)  
    print("Your loss was correct for a test point")  
except:  
    print("Your loss was not correct on a test point")

Your loss was correct for a test point

# 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.

[ ]

Start coding or generate with AI.

keyboard\_arrow\_down

## 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]

[ ]

# Part 1  
  
# Fill in the answer to part 1 below as a decimal number  
problem1\_p1 = 0.28

[ ]

# Part 2  
  
# Fill in the answer to part 2 below as a decimal number  
problem1\_p2 = 0.22

[ ]

# Part 3  
  
# Fill in the answer to part 3 below as a boolean  
problem1\_irreducible = True

[ ]

# 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 = ([0.28888889, 0.41111111, 0.3])

[ ]

# Part 5  
  
# Fill in the answer to part 5 below  
# That is, the expected number of steps as a decimal number  
problem1\_ET = 3.844520365145998

keyboard\_arrow\_down

## Assignment 2, PROBLEM 2

Maximum Points = 4

Use the **Multi-dimensional Constrained Optimisation** example (in 07-Optimization.ipynb) to numerically find the MLe for the mean and variance parameter based on normallySimulatedDataSamples, an array obtained by a specific simulation of 30 IID samples from the Normal(10,2) random variable.

Recall that Normal(μ,σ2) RV has the probability density function given by:

f(x;μ,σ)=1σ2π−−√exp(−12σ2(x−μ)2)

The two parameters, μ∈R:=(−∞,∞) and σ∈(0,∞), are sometimes referred to as the location and scale parameters.

You know that the log likelihood function for n IID samples from a Normal RV with parameters μ and σ simply follows from ∑ni=1log(f(xi;μ,σ)), based on the IID assumption.

NOTE: When setting bounding boxes for μ and σ try to start with some guesses like [−20,20] and [0.1,5.0] and make it larger if the solution is at the boundary. Making the left bounding-point for σ too close to 0.0 will cause division by zero Warnings. Other numerical instabilities can happen in such iterative numerical solutions to the MLe. You need to be patient and learn by trial-and-error. You will see the mathematical theory in more details in a future course in scientific computing/optimisation. So don't worry too much now except learning to use it for our problems.

[ ]

import numpy as np  
from scipy import optimize  
  
# Set random seed for reproducibility  
np.random.seed(123456)  
  
# Simulate 30 IID samples drawn from Normal(10, 2) RV  
normallySimulatedDataSamples = np.random.normal(10, 2, 30)  
  
# Define the negative log-likelihood function to minimize  
def neg\_log\_likelihood(parameters):  
    """Return the negative log-likelihood of normallySimulatedDataSamples with mean and variance parameters."""  
    mu\_param, sigma\_param = parameters  
  
    # Avoid non-positive sigma values to prevent division by zero  
    if sigma\_param <= 0:  
        return np.inf  
  
    # Calculate the negative log-likelihood  
    n = len(normallySimulatedDataSamples)  
    log\_likelihood = -n \* np.log(sigma\_param \* np.sqrt(2 \* np.pi)) - np.sum((normallySimulatedDataSamples - mu\_param) \*\* 2) / (2 \* sigma\_param \*\* 2)  
  
    return -log\_likelihood  # Return negative log-likelihood for minimization  
  
# Specify bounds and initial guess  
parameter\_bounding\_box = ((-20, 20), (0.1, 5.0))  # Bounds for mu and sigma  
initial\_guess = np.array([10, 2])  # Initial guess for mu and sigma  
  
# Perform the optimization  
result\_problem2\_opt = optimize.minimize(neg\_log\_likelihood, initial\_guess, bounds=parameter\_bounding\_box, method='L-BFGS-B')  
  
# Display the result  
print("Optimization Result:", result\_problem2\_opt)  
print("MLE for the mean (μ):", result\_problem2\_opt.x[0])  
print("MLE for the standard deviation (σ):", result\_problem2\_opt.x[1])

Optimization Result: message: CONVERGENCE: REL\_REDUCTION\_OF\_F\_<=\_FACTR\*EPSMCH

success: True

status: 0

fun: 58.631387282482464

x: [ 9.269e+00 1.708e+00]

nit: 5

jac: [ 9.237e-06 -6.821e-05]

nfev: 27

njev: 9

hess\_inv: <2x2 LbfgsInvHessProduct with dtype=float64>

MLE for the mean (μ): 9.26862051786077

MLE for the standard deviation (σ): 1.7081981772959267

keyboard\_arrow\_down

## Assignment 2, PROBLEM 3

Maximum Points = 4

Derive the maximum likelihood estimate for n IID samples from a random variable with the following probability density function:

f(x;λ)=124λ5x4exp(−λx), where, λ>0,x>0

You can solve the MLe by hand (using pencil paper or using key-strokes). Present your solution as the return value of a function called def MLeForAssignment2Problem3(x), where x is a list of n input data points.

[ ]

# do not change the name of the function, just replace XXX with the appropriate expressions for the MLe  
def MLeForAssignment2Problem3(x):  
    '''write comment of what this function does'''  
    n = len(x)  
    lambda\_mle = 5 \* n / sum(x)  
    return lambda\_mle

## Assignment 2, PROBLEM 4

Maximum Points = 8

keyboard\_arrow\_down

## 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

p0(x)=π2|sin(2πx)|,x∈[0,1].

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

[ ]

def problem4\_LCG(size=None, seed=0):  
    """  
    A linear congruential generator that generates pseudo random numbers 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  
    """  
    # Parameters for the LCG that satisfy the Hull-Dobell Theorem  
    a = 16807               # Multiplier  
    b = 0                   # Increment  
    M = 2\*\*31 - 1           # Modulus, a large prime number for a full period  
  
    # Initialize the random number list and the seed  
    out = []  
    current = seed  
  
    # Generate the sequence of pseudo-random numbers  
    for \_ in range(size):  
        current = (a \* current + b) % M  
        out.append(current / M)  # Normalize to the range [0, 1]  
  
    return out

[ ]

def problem4\_uniform(generator=None, period=1, size=None, seed=0):  
    """  
    Takes a generator and produces samples from the uniform [0,1] distribution according  
    to size.  
  
    Parameters  
    -------------  
    generator : a function of type generator(size, seed) and produces the same result as problem1\_LCG, i.e. pseudo-random numbers in the 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  
    """  
  
               # Modulus  
  
    # Generate the raw samples using the generator  
    raw\_samples = generator(size=size, seed=seed)  
  
    # Convert the raw samples into the range [0,1)  
    uniform\_samples = [x/(period-1) for x in raw\_samples]  
  
    return uniform\_samples

[ ]

import math  
  
def problem4\_accept\_reject(uniformGenerator=None, n\_samples=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 or until n\_samples accepted samples are obtained.  
  
    Parameters  
    -------------  
    uniformGenerator : a function of type generator(size, seed) that produces uniform pseudo-random  
    numbers from [0,1]  
    n\_samples : an integer denoting how many accepted samples should be produced  
    n\_iterations : an integer denoting the maximum number of attempts in the accept-reject sampler  
    seed : the seed to be used in the generator provided  
  
    Returns  
    --------------  
    out : a list of the pseudo-random numbers with the specified distribution  
    """  
  
    # Define the target distribution function (p(x) = (pi/2) \* |sin(2\*pi\*x)|)  
    def target\_distribution(x):  
        return (math.pi / 2) \* abs(math.sin(2 \* math.pi \* x))  
  
    # Define the upper bound of the target distribution M  
    M = math.pi / 2  # Since sin(x) is between -1 and 1, the maximum value of |sin(x)| is 1  
  
    accepted\_samples = []  
  
    # Run the Accept-Reject sampler for n\_iterations or until n\_samples are accepted  
    iterations = 0  
    while len(accepted\_samples) < n\_samples and iterations < n\_iterations:  
        # Generate a uniform random number (candidate) from the uniform generator  
        x = uniformGenerator(size=1, seed=seed + iterations)[0]  
  
        # Generate a uniform random number for accept/reject decision  
        u = uniformGenerator(size=1, seed=seed + iterations + 1)[0]  
  
        # Calculate the target distribution value at x  
        target\_value = target\_distribution(x)  
  
        # Accept the sample if u is less than or equal to target\_value / M  
        if u <= target\_value / M:  
            accepted\_samples.append(x)  
  
        iterations += 1  
  
    return accepted\_samples

keyboard\_arrow\_down

#### Local Test for Assignment 2, PROBLEM 4

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.

[ ]

# If you managed to solve all three parts you can test the following code to see if it runs  
# you have to change the period to match your LCG though, this is marked 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" % problem4\_LCG(size=10, seed = 1))  
  
period = 32  
  
print("Uniform sampler %s" % problem4\_uniform(generator=problem4\_LCG, period = period, size=10, seed=1))  
  
uniform\_sampler = lambda size,seed: problem4\_uniform(generator=problem4\_LCG, period = period, size=size, seed=seed)  
  
print("Accept-Reject sampler %s" % problem4\_accept\_reject(uniformGenerator = uniform\_sampler,n\_samples=10,n\_iterations=20,seed=1))

LCG output: [7.826369259425611e-06, 0.13153778814316625, 0.7556053221950332, 0.4586501319234493, 0.5327672374121692, 0.21895918632809036, 0.04704461621448613, 0.678864716868319, 0.6792964058366122, 0.9346928959408276]

Uniform sampler [2.5246352449760035e-07, 0.004243154456231169, 0.024374365232097843, 0.014795165545917718, 0.017186039916521588, 0.007063199558970657, 0.0015175682649834234, 0.0218988618344619, 0.021912787285052006, 0.030151383740026697]

Accept-Reject sampler [2.5246352449760035e-07, 5.049270489952007e-07, 7.573905734928011e-07, 1.0098540979904014e-06, 1.2623176224880017e-06, 1.5147811469856021e-06, 1.7672446714832024e-06, 2.019708195980803e-06, 2.2721717204784033e-06, 2.5246352449760033e-06]

[ ]

# 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" % problem4\_accept\_reject(uniformGenerator=testUniformGenerator,n\_samples=10, n\_iterations=20, seed=1))



Accept-Reject sampler [0.9560342718892494, 0.23796462709189137, 0.23604808973743452, 0.793340083761663, 0.32383276483316237, 0.2267058593810488, 0.2590084917154736, 0.36152277491407514, 0.18126486333322134, 0.9056396761745207]

# Assignment 1 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.

keyboard\_arrow\_down

## Assignment 1, PROBLEM 1

Maximum Points = 3

Given that you are being introduced to data science it is important to bear in mind the true costs of AI, a highly predictive family of algorithms used in data engineering sciences:

Read the 16 pages of [ai-anatomy-publication.pdf](https://www.google.com/url?q=http%3A%2F%2Fwww.anatomyof.ai%2Fimg%2Fai-anatomy-publication.pdf) with the highly detailed [ai-anatomy-map.pdf](https://www.google.com/url?q=https%3A%2F%2Fanatomyof.ai%2Fimg%2Fai-anatomy-map.pdf) of [https://anatomyof.ai/](https://www.google.com/url?q=https%3A%2F%2Fanatomyof.ai%2F), "Anatomy of an AI System" By Kate Crawford and Vladan Joler (2018). The first problem in ASSIGNMENT 1 is a trivial test of your reading comprehension.

Answer whether each of the following statements is True or False according to the authors by appropriately replacing Xxxxx coresponding to TruthValueOfStatement0a, TruthValueOfStatement0b and TruthValueOfStatement0c, respectively, in the next cell to demonstrate your reading comprehension.

1. Statement0a = Each small moment of convenience (provided by Amazon's Echo) – be it answering a question, turning on a light, or playing a song – requires a vast planetary network, fueled by the extraction of non-renewable materials, labor, and data.
2. Statement0b = The Echo user is simultaneously a consumer, a resource, a worker, and a product
3. Statement0c = Many of the assumptions about human life made by machine learning systems are narrow, normative and laden with error. Yet they are inscribing and building those assumptions into a new world, and will increasingly play a role in how opportunities, wealth, and knowledge are distributed.

[ ]

# Replace Xxxxx with True or False; Don't modify anything else in this cell!  
  
TruthValueOfStatement0a = True  
  
TruthValueOfStatement0b = True  
  
TruthValueOfStatement0c = True

keyboard\_arrow\_down

#### Local Test for Assignment 1, PROBLEM 1

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.

[ ]

# Test locally to ensure an acceptable answer, True or False  
try:  
    assert(isinstance(TruthValueOfStatement0a, bool))  
    assert(isinstance(TruthValueOfStatement0b, bool))  
    assert(isinstance(TruthValueOfStatement0c, bool))  
except:  
    print("Try again. You are not writing True or False for your answers.")  
else:  
    print("Good, you have answered either True or False. Hopefully they are the correct answers!")

Good, you have answered either True or False. Hopefully they are the correct answers!

keyboard\_arrow\_down

## Assignment 1, PROBLEM 2

Maximum Points = 2

Evaluate the following cells by replacing X with the right command-line option to head in order to find the first four lines of the csv file data/final.csv

%%sh  
man head  
  
HEAD(1)                   BSD General Commands Manual                  HEAD(1)  
  
NAME  
     head -- display first lines of a file  
  
SYNOPSIS  
     head [-n count | -c bytes] [file ...]  
  
DESCRIPTION  
     This filter displays the first count lines or bytes of each of the speci-  
     fied files, or of the standard input if no files are specified.  If count  
     is omitted it defaults to 10.  
  
     If more than a single file is specified, each file is preceded by a  
     header consisting of the string ``==> XXX <=='' where ``XXX'' is the name  
     of the file.  
  
EXIT STATUS  
     The head utility exits 0 on success, and >0 if an error occurs.  
  
SEE ALSO  
     tail(1)  
  
HISTORY  
     The head command appeared in PWB UNIX.  
  
BSD                              June 6, 1993                              BSD

[ ]

%%sh  
head -n 4 data/final.csv

region,municipality,district,party,votes

Blekinge län,Karlshamn,0 - Centrala Asarum,S,519

Blekinge län,Karlshamn,0 - Centrala Asarum,SD,311

Blekinge län,Karlshamn,0 - Centrala Asarum,M,162

[ ]

line\_1\_final = "region,municipality,district,party,votes"  
line\_2\_final = "Blekinge län,Karlshamn,0 - Centrala Asarum,S,519"

keyboard\_arrow\_down

#### Local Test for Assignment 1, PROBLEM 2

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.

[ ]

# Evaluate this cell locally to make sure you have the answer as a string  
try:  
    assert(type(line\_1\_final) == str)  
    print("Good! You have answered as a string for line 1. Hopefully it is the correct!")  
except AssertionError:  
    print("Try Again. You should answer with a string.")  
try:  
    assert(type(line\_2\_final) == str)  
    print("Good! You have answered as a string for line 2. Hopefully it is the correct!")  
except AssertionError:  
    print("Try Again. You should answer with a string.")

Good! You have answered as a string for line 1. Hopefully it is the correct!

Good! You have answered as a string for line 2. Hopefully it is the correct!

keyboard\_arrow\_down

## Assignment 1, PROBLEM 3

Maximum Points = 3

In this assignment the goal is to parse the final.csv file from the previous problem.

1. Read the file data/final.csv and parse it using the csv package and store the result as follows

the header variable contains a list of names all as strings

the data variable should be a list of lists containing all the rows of the csv file

[ ]

#import pandas as pds  
  
  
#file\_path = 'final.csv'  
  
  
#df = pds.read\_csv(file\_path)  
  
  
#header = list(df.columns)  
  
  
#data = df.values.tolist()  
  
  
#print("Header:", header)  
#print("First 5 rows of data:", data[:5])  
  
import csv  
  
with open('data/final.csv', mode='r') as f:  
    reader = csv.reader(f)  
    header = next(reader)  
    data = []  
    for row in reader:  
        data.append(row)  
print(header)  
print(data[:5])

['region', 'municipality', 'district', 'party', 'votes']

[['Blekinge län', 'Karlshamn', '0 - Centrala Asarum', 'S', '519'], ['Blekinge län', 'Karlshamn', '0 - Centrala Asarum', 'SD', '311'], ['Blekinge län', 'Karlshamn', '0 - Centrala Asarum', 'M', '162'], ['Blekinge län', 'Karlshamn', '0 - Centrala Asarum', 'V', '82'], ['Blekinge län', 'Karlshamn', '0 - Centrala Asarum', 'KD', '53']]

keyboard\_arrow\_down

#### Local Test for Assignment 1, 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.

[ ]

# Evaluate this cell locally to make sure you have the answer in the right format  
try:  
    assert(type(header) == list)  
    print("Good! You have the header as a list. Hopefully it is the correct!")  
except AssertionError:  
    print("Try Again. You should answer with a list.")  
try:  
    types = set([type(a) for a in header])  
    assert((len(types) == 1) and (list(types)[0] == str))  
    print("Good! You have the header as a list of strings. Hopefully it is the correct!")  
except AssertionError:  
    print("Try Again. You should answer with a list of strings.")  
try:  
    assert(type(data) == list)  
    print("Good! You have the data as a list. Hopefully it is the correct!")  
except AssertionError:  
    print("Try Again. You should answer with a list.")  
try:  
    types = set([type(a) for a in data])  
    assert((len(types) == 1) and (list(types)[0] == list))  
    print("Good! You have the data as a list of lists. Hopefully it is the correct!")  
except AssertionError:  
    print("Try Again. You should answer with a list of lists.")  
try:  
    types = set(sum([[type(d) for d in t] for t in data[:1]],[]))  
    assert((len(types) == 1) and (list(types)[0] == str))  
    print("Good! You have the data as a list of lists of strings. Hopefully it is the correct!")  
except AssertionError:  
    print("Try Again. You should answer with a list of lists of strings.")

Good! You have the header as a list. Hopefully it is the correct!

Good! You have the header as a list of strings. Hopefully it is the correct!

Good! You have the data as a list. Hopefully it is the correct!

Good! You have the data as a list of lists. Hopefully it is the correct!

Good! You have the data as a list of lists of strings. Hopefully it is the correct!

## Assignment 1, PROBLEM 4

Maximum Points = 8

keyboard\_arrow\_down

## Students passing exam (Sample exam problem)

Let's say we have an exam question which consists of 10 yes/no questions. From past performance of similar students, a randomly chosen student will know the correct answer to N∼binom(10,6/10) questions. Furthermore, we assume that the student will guess the answer with equal probability to each question they don't know the answer to, i.e. given N we define Z∼binom(10−N,1/2) as the number of correctly guessed answers. Define Y=N+Z, i.e., Y represents the number of total correct answers.

We are interested in setting a deterministic threshold T, i.e., we would pass a student at threshold T if Y≥T. Here T∈{0,1,2,…,10}.

1. [5p] For each threshold T, compute the probability that the student knows less than 5 correct answers given that the student passed, i.e., N<5. Put the answer in problem11\_probabilities as a list.
2. [3p] What is the smallest value of T such that if Y≥T then we are 90% certain that N≥5?

[ ]

# Hint the PMF of N is p\_N(k) where p\_N is  
from scipy.special import binom as binomial  
p = 6/10  
p\_N = lambda k: binomial(10,k)\*((1-p)\*\*(10-k))\*((p)\*\*k)

[ ]

# Part 1:  
# replace XXX to represent P(N < 5) for T = [0,1,2,...,10], i.e. your answer should be a list  
# of length 11.  
problem11\_probabilities = [0.16623861760000014, 0.16623853222282584, 0.16623511712151587, 0.16617364051358025, 0.1655173254906335, 0.16089403080809428, 0.14445279891451662, 0.11223025722358074, 0.07321212315323337, 0.04058456420898444, 0.019727706909179715]

[ ]

# Part 2: Give an integer between 0 and 10 which is the answer to 2.  
problem12\_T = 8

## Assignment 1, PROBLEM 5

Maximum Points = 8

keyboard\_arrow\_down

## Concentration of measure (Sample exam problem)

As you recall, we said that concentration of measure was simply the phenomenon where we expect that the probability of a large deviation of some quantity becoming smaller as we observe more samples: [0.4 points per correct answer]

1. Which of the following will exponentially concentrate, i.e. for some C1,C2,C3,C4

P(Z−E[Z]≥ϵ)≤C1e−C2nϵ2∨C3e−C4n(ϵ+1).

* 1. The empirical variance of i.i.d. random variables with finite mean?
  2. The empirical variance of i.i.d. sub-Gaussian random variables?
  3. The empirical variance of i.i.d. sub-Exponential random variables?
  4. The empirical mean of i.i.d. sub-Gaussian random variables?
  5. The empirical mean of i.i.d. sub-Exponential random variables?
  6. The empirical mean of i.i.d. random variables with finite variance?
  7. The empirical third moment of i.i.d. random variables with finite sixth moment?
  8. The empirical fourth moment of i.i.d. sub-Gaussian random variables?
  9. The empirical mean of i.i.d. deterministic random variables?
  10. The empirical tenth moment of i.i.d. Bernoulli random variables?

1. Which of the above will concentrate in the weaker sense, that for some C1

P(Z−E[Z]≥ϵ)≤C1nϵ2?

[ ]

# Answers to part 1, which of the alternatives exponentially concentrate, answer as a list

# i.e. [1,4,5] that is example 1, 4, and 5 concentrate

problem3\_answer\_1 = [2,4,5,8,9,10]



[ ]

# Answers to part 2, which of the alternatives concentrate in the weaker sense, answer as a list

# i.e. [1,4,5] that is example 1, 4, and 5 concentrate

problem3\_answer\_2 = [2,3,4,5,6,7,8,9,10]

