CMSC 173 - MP 2

Instructions:

1. Create an overview of the problem being solved, e.g., what was the story behind the collection of the data, description of the attributes/features used,etc.
2. (Data Preprocessing and Exploratory Analysis) Present descriptive statistics as applicable (e.g., distribution, central tendency, variability) of the data before training the models. Clean the data if there are missing values, etc. You may perform feature engineering (i.e., creating new features out of the given features), but be sure to document your justifications.
3. Split your data into proportions of 70% training set and 30% testing set.
4. Train the following models: (a) logistic regression classifier and (b) naive Bayes classifier on the dataset.
5. Evaluate the performance of the trained model. You may use additional performance measures if you want, but for now I will only require the calculation of the accuracy. The accuracy measures the fraction of correct classifications. With this, you need to generate the confusion matrix. You may read this if you haven't encountered this concept before: https://www.sciencedirect.com/topics/engineering/confusion-matrix#:~:text=A%20confusion%20matrix%20represents%20the,by%20model%20as%20oth Remember to compute this matrix from the test set (not the training set).

In [ ]:
```
using Random
using StatsBase
using CSV
using DataFrames
using Plots
using Base
```

In [ ]:
```
dataset = CSV.read("passenger_flight.csv",DataFrame)
Random.seed!(123)
dataset = dataset[shuffle(axes(dataset, 1)), :]
```

Out[ ]: 25976×23 DataFrame *25951 rows omitted*

Row	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	E C b
	Int64	Int64	Int64	Int64	Int64	Int64	Int64	Int64	In
1	1	1	50	1	1	3744	5	5	
2	0	1	53	1	1	2661	4	5	
3	1	1	20	0	0	541	2	4	
4	0	1	52	0	1	944	1	2	
5	1	1	33	1	1	406	1	1	
6	0	1	51	0	0	621	2	4	
7	1	1	25	1	1	3547	2	2	
8	0	1	51	1	1	547	4	4	
9	0	1	60	0	1	438	2	4	
10	1	1	26	1	1	2085	1	1	
11	1	1	17	0	0	505	3	4	
12	0	0	22	1	0	329	3	1	
13	1	1	25	0	0	479	3	4	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	
25965	0	1	27	1	1	1716	2	1	
25966	1	0	26	1	1	591	1	1	
25967	0	1	63	0	0	1024	4	4	
25968	1	1	39	1	1	2131	2	2	
25969	0	1	40	0	0	369	3	1	
25970	1	1	38	0	0	633	2	1	
25971	1	1	32	1	1	1635	2	2	
25972	0	1	43	1	1	1055	5	5	
25973	1	1	61	1	1	2273	4	4	
25974	1	1	37	1	1	695	2	4	
25975	0	1	38	0	0	1313	4	5	
25976	1	0	26	1	1	447	1	0	

Data Preprocessing

In [ ]:
```
# REMOVE MISSING
has_missing = .!completecases(dataset)

# check rows with missing values
rows_with_missing_values = dataset[has_missing, :] # 83 rows have missing values in

# remove missing values since it is difficult to fill the missing values
dataset = dataset[.!has_missing, :]

# rename column names
col_names = names(dataset)
new_col_names = map(lowercase, String.(col_names)) # convert to lower case
new_col_names .= replace.(new_col_names, " "=>"_", "-"=>"", "/"=>"_") # replace spa
rename!(dataset, new_col_names)
```

Out[ ]: 25893×23 DataFrame *25868 rows omitted*

Row	gender	customer_type	age	type_of_travel	class	flight_distance	inflight_wifi
	Int64	Int64	Int64	Int64	Int64	Int64	Int64
1	1	1	50	1	1	3744	
2	0	1	53	1	1	2661	
3	1	1	20	0	0	541	
4	0	1	52	0	1	944	
5	1	1	33	1	1	406	
6	0	1	51	0	0	621	
7	1	1	25	1	1	3547	
8	0	1	51	1	1	547	
9	0	1	60	0	1	438	
10	1	1	26	1	1	2085	
11	1	1	17	0	0	505	
12	0	0	22	1	0	329	
13	1	1	25	0	0	479	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	
25882	1	1	39	0	0	1476	
25883	0	1	27	1	1	1716	
25884	1	0	26	1	1	591	
25885	0	1	63	0	0	1024	
25886	1	1	39	1	1	2131	
25887	0	1	40	0	0	369	
25888	1	1	38	0	0	633	
25889	1	1	32	1	1	1635	
25890	0	1	43	1	1	1055	
25891	1	1	61	1	1	2273	
25892	1	1	37	1	1	695	
25893	0	1	38	0	0	1313	

In [ ]:

```
# split dataframe into 2 df depending on pct
function splitdf(df, pct)
```

```
    @assert 0 <= pct <= 1
    ids = collect(axes(df, 1))
    shuffle!(ids)
    sel = ids .<= nrow(df) .* pct
    train = view(df, sel, :)
    test = view(df, .!sel, :)

    # println(hcat(train[:,1:end-1], DataFrame("satisfaction"=>train[:,end])) == tr

    return train[:,1:end-1], DataFrame("satisfaction"=>train[:,end]), test[:,1:end-
end

(x_train, y_train, x_test, y_test) = splitdf(dataset, 0.7)
```

Out[ ]: (18125×22 DataFrame

Row	gender	customer_type	age	type_of_travel	class	flight_distance	⋯
	Int64	Int64	Int64	Int64	Int64	Int64	⋯
1	1	1	50	1	1	3744	⋯
2	0	1	52	0	1	944	
3	1	1	33	1	1	406	
4	1	1	25	1	1	3547	
5	0	1	51	1	1	547	⋯
6	0	1	60	0	1	438	
7	1	1	26	1	1	2085	
8	1	1	17	0	0	505	
9	0	0	22	1	0	329	⋯
10	1	1	25	0	0	479	
11	0	1	24	0	0	678	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱
18116	0	1	31	1	1	1136	
18117	0	1	53	0	0	224	⋯
18118	1	0	56	1	0	132	
18119	0	1	32	1	1	1269	
18120	1	1	39	0	0	1476	
18121	0	1	27	1	1	1716	⋯
18122	1	0	26	1	1	591	
18123	0	1	63	0	0	1024	
18124	0	1	40	0	0	369	
18125	1	1	32	1	1	1635	⋯

16 columns and 18104 rows omitted,

18125×1 DataFrame

Row	satisfaction
	Int64
1	1
2	0
3	1
4	1
5	1
6	0
7	1
8	0
9	0
10	0
11	0
⋮	⋮
18116	1
18117	0
18118	0
18119	0
18120	0
18121	0
18122	0
18123	1
18124	0
18125	1

18104 rows omitted, 7768×22 DataFrame

Row	gender	customer_type	age	type_of_travel	class	flight_distance	⋯
	Int64	Int64	Int64	Int64	Int64	Int64	⋯

```
─────┼──────────────────────────────────────────────────────────────────────────────
   1 │     0          1    53              1      1            2661  ⋯
   2 │     1          1    20              0      0             541
   3 │     0          1    51              0      0             621
   4 │     0          1    47              0      0             483
   5 │     1          0    28              1      0             731  ⋯
   6 │     0          1    27              1      1            2380
   7 │     1          1    66              0      0            3904
   8 │     1          1    63              0      0             622
   9 │     0          0    35              1      0            1010  ⋯
  10 │     0          0    21              1      0             408
  11 │     0          1    17              0      1             912
  ⋮  │  ⋮         ⋮     ⋮          ⋮       ⋮         ⋮          ⋱
7759 │     1          1    68              1      1            1678
7760 │     1          1    53              1      1            3435  ⋯
7761 │     1          1    36              1      0             190
7762 │     0          1    57              1      1            1197
7763 │     1          1    39              1      1            2131
7764 │     1          1    38              0      0             633  ⋯
7765 │     0          1    43              1      1            1055
7766 │     1          1    61              1      1            2273
7767 │     1          1    37              1      1             695
7768 │     0          1    38              0      0            1313  ⋯
                                          16 columns and 7747 rows omitted,
7768×1 DataFrame
  Row │ satisfaction
      │ Int64
──────┼──────────────
    1 │            0
    2 │            0
    3 │            0
    4 │            0
    5 │            0
    6 │            0
    7 │            0
    8 │            0
    9 │            0
   10 │            1
   11 │            0
  ⋮   │      ⋮
 7759 │            1
 7760 │            1
 7761 │            1
 7762 │            1
 7763 │            1
 7764 │            0
 7765 │            1
 7766 │            1
 7767 │            0
 7768 │            1
    7747 rows omitted)
```

Naive Bayes

In [ ]:
```
# build conditional probability table

cont_col_names = ["age", "flight_distance", "departure_delay_in_minutes", "arrival_
disc_col_names = [name for name in names(dataset) if name ∉ cont_col_names && name≠
train = hcat(x_train, y_train)

# calculate discrete probabilities
function count_disc_prob(df, col_name)
    return combine(groupby(df, [col_name, "satisfaction"]), nrow)
end

cond_prob_table = Dict()
for name in disc_col_names
    cond_prob_table[name] = count_disc_prob(train, name)
end

# calculate continuous probabilities
function count_cont_prob(df, col_name)
    a = combine(groupby(df, "satisfaction"), [col_name] => mean, [col_name] => std)
end

for name in cont_col_names
    cond_prob_table[name] = count_cont_prob(train, name)
end
```

```
2×3 DataFrame
 Row │ satisfaction  age_mean  age_std
     │ Int64         Float64   Float64
─────┼─────────────────────────────────
   1 │            0   38.0589  16.5255
   2 │            1   41.6782  12.955
```

In [ ]:
```
# calculate likelihood for continuous data
function likelihood(cond_prob_table, feature, satisfaction, x)
    feature_table = cond_prob_table[feature]
    prob_values = filter(row -> row.satisfaction == satisfaction, feature_table)

    # get mean and variance
    μ = prob_values[1,2]
    σ = prob_values[1,3]

    return (1/(σ * sqrt(2π))) * exp((-1/2) * ((x-μ)/σ)^2)
end

# calculate probabilities for discrete (categorical) data
function disc_cond_prob(cond_prob_table, feature, satisfaction, x)
    feature_table = cond_prob_table[feature]
    feature_table = filter(row -> row.satisfaction==satisfaction, feature_table)
    total = sum(feature_table[:,:nrow])

    val = 0
    try
        val = filter(row -> row[feature] == x, feature_table)[1,end]
    catch
        val = 0
    end
```

```
    # apply laplace smoothing
    return (val+1)/(total+1)
end

# run test
function test()
    correct = 0
    not_correct = 0

    # iterate all training data
    for i in 1:size(x_train)[1]
        test_case = x_train[i,:]
        p_satisfied_proportional = 1
        p_not_satisfied_proportional = 1

        # get probabilities of all features
        for col_name in names(test_case)

            # treat discrete and continuous features separately
            if col_name ∈ disc_col_names
                p_satisfied_proportional *= disc_cond_prob(cond_prob_table, col_nam
                p_not_satisfied_proportional *= disc_cond_prob(cond_prob_table, col
            else
                p_satisfied_proportional *= likelihood(cond_prob_table, col_name, 1
                p_not_satisfied_proportional *= likelihood(cond_prob_table, col_nam
            end
        end

        # calculate probabilities
        p_satisfied = (p_satisfied_proportional / (p_satisfied_proportional+p_not_s
        p_not_satisfied = (p_not_satisfied_proportional / (p_satisfied_proportional

        # count correct and incorrect predictions
        if (p_satisfied > p_not_satisfied && y_train[i,1] == 1) || (p_satisfied < p
            correct += 1
        else
            not_correct += 1
        end
    end

    println("Correct predictions: ", correct)
    println("Incorrect predictions: ", not_correct)
    println("Accuracy: ", (correct / (correct + not_correct))*100)
end

test()
```

```
Correct predictions: 15924
Incorrect predictions: 2201
Accuracy: 87.85655172413793
```

In [ ]: