Contents

[List of Figures 3](#_Toc151411574)

[Part 1: Load Dataset 4](#_Toc151411575)

[Solution 4](#_Toc151411576)

[Code 4](#_Toc151411577)

[Extra 1: Dataset Snippet 4](#_Toc151411578)

[Extra 2: Dataset Statistics 4](#_Toc151411579)

[Part 2: Stratified Sampling 5](#_Toc151411580)

[Solution 5](#_Toc151411581)

[Extra 1: Confirmation of 75/25 split 5](#_Toc151411582)

[Extra 2: Confirmation of stratification 5](#_Toc151411583)

[Code: 5](#_Toc151411584)

[Part 3: Visualization 6](#_Toc151411585)

[Solution: 6](#_Toc151411586)

[Extra 1: Histogram for bin count = 10 8](#_Toc151411587)

[Extra 2: Density Plots 9](#_Toc151411588)

[Extra 3: Worst Feature 11](#_Toc151411589)

[Code: 11](#_Toc151411590)

[Part 4: Training with Cross Validation: 12](#_Toc151411591)

[Solution: 12](#_Toc151411592)

[Extra 1: Stratified Cross Validation 12](#_Toc151411593)

[Extra 2: Visualization for without stratification 13](#_Toc151411594)

[Extra 3: Visualization for with stratification 13](#_Toc151411595)

[Code 13](#_Toc151411596)

[Part 5: Training without Cross Validation 15](#_Toc151411597)

[Solution 15](#_Toc151411598)

[Extra 1: Note form the lectures 15](#_Toc151411599)

[Extra 2: Visualization 15](#_Toc151411600)

[Code: 15](#_Toc151411601)

[Part 6: Misclassification Cost 16](#_Toc151411602)

[Solution: 16](#_Toc151411603)

[Extra 0: Alternative for incorporating the loss 17](#_Toc151411604)

[Extra 1: Misclassification Cost Calculation 17](#_Toc151411605)

[Extra 2: Hinge Loss Formula 17](#_Toc151411606)

[Extra 3: Multi class classification for SVM 17](#_Toc151411607)

[Extra 4: References 17](#_Toc151411608)

[Code 17](#_Toc151411609)

[Part 7: Neural Networks (NN) 18](#_Toc151411610)

[Solution: 18](#_Toc151411611)

[Extra 1: Hyperparameters for the NN model 18](#_Toc151411612)

[Extra 2: Plots for training Losses and Accuracy for each epoch 18](#_Toc151411613)

[Code: 19](#_Toc151411614)

[Part 8: Naïve Bayes Classifier 22](#_Toc151411615)

[Solution: 22](#_Toc151411616)

[Extra 1: Significance Level (alpha) 22](#_Toc151411617)

[Extra 2: Number of iterations 22](#_Toc151411618)

[Extra 3: Distribution of mean under H0 23](#_Toc151411619)

[Code 23](#_Toc151411620)

[Appendix: 25](#_Toc151411621)

[Unimportant code: 25](#_Toc151411622)

# List of Figures

[Figure 1: Dataset Snippet 4](#_Toc151410564)

[Figure 2: Dataset Statistics 4](#_Toc151410565)

[Figure 3: Histogram of distribution for different Efficiency classes for the feature Thermal 6](#_Toc151410566)

[Figure 4: Histogram of distribution for different Efficiency classes for the feature Area 6](#_Toc151410567)

[Figure 5: Histogram of distribution for different Efficiency classes for the feature Glazing 7](#_Toc151410568)

[Figure 6: Histogram of distribution for different Efficiency classes for the feature Cladding 7](#_Toc151410569)

[Figure 7: Histogram of distribution for different Efficiency classes for the feature Roofing 7](#_Toc151410570)

[Figure 8: Histogram with bin count 10 of distribution for different Efficiency classes for the feature Thermal 8](#_Toc151410571)

[Figure 9: Histogram with bin count 10 of distribution for different Efficiency classes for the feature Area 8](#_Toc151410572)

[Figure 10: Histogram with bin count 10 of distribution for different Efficiency classes for the feature Glazing 8](#_Toc151410573)

[Figure 11:Histogram with bin count 10 of distribution for different Efficiency classes for the feature Cladding 9](#_Toc151410574)

[Figure 12: Histogram with bin count 10 of distribution for different Efficiency classes for the feature Roofing 9](#_Toc151410575)

[Figure 13: Density plot for different efficiency classes for the feature Thermal 9](#_Toc151410576)

[Figure 14: Density plot for different efficiency classes for the feature Area 10](#_Toc151410577)

[Figure 15: Density plot for different efficiency classes for the feature Glazing 10](#_Toc151410578)

[Figure 16: Density plot for different efficiency classes for the feature Cladding 10](#_Toc151410579)

[Figure 17: Density plot for different efficiency classes for the feature Roofing 11](#_Toc151410580)

[Figure 18: Accuracy plot for folds without stratified Cross validation. 13](#_Toc151410581)

[Figure 19: Accuracy plot for folds with stratified Cross validation. 13](#_Toc151410582)

[Figure 20: Accuracy visualization with standard deviation 15](#_Toc151410583)

[Figure 21: Accuracy and loss for 1 hidden neuron in each fold 18](#_Toc151410584)

[Figure 22: Accuracy and loss for 10 hidden neurons in each fold 19](#_Toc151410585)

[Figure 23: Accuracy and loss for 100 hidden neurons in each fold 19](#_Toc151410586)

[Figure 24: Distribution of Mean under H0 23](#_Toc151410587)

# Part 1: Load Dataset

Load the dataset in A4.txt. The column names correspond to the five features plus the class ID: colNames = ['Thermal', 'Area', 'Glazing', 'Clading', 'Roofing', 'Efficiency']

## Solution

### Code

# init

COLUMNS = ['Thermal', 'Area', 'Glazing', 'Clading', 'Roofing', 'Efficiency']

DATA\_PATH = "../data/A4.csv"

# read dataset

dataset = pd.read\_csv(

    DATA\_PATH,

    header=None,

    )

dataset.columns = COLUMNS

### Extra 1: Dataset Snippet

A screenshot of a black screen

Description automatically generated

Figure : Dataset Snippet

### Extra 2: Dataset Statistics

A screenshot of a computer screen

Description automatically generated

Figure : Dataset Statistics

# Part 2: Stratified Sampling

Split your data into train/test using a 75/25 split and stratified sampling. Report the number of samples from each class in your train and test subsets.

## Solution

|  |  |  |  |
| --- | --- | --- | --- |
| Class | Train Sample Count | Test Sample Count | Complete dataset Count |
| Low=0 | 131 | 44 | 175 |
| Med=1 | 68 | 22 | 90 |
| High=2 | 26 | 9 | 35 |

Table : Sample count for each set of data

Table 1 contains the number of samples for each class (low, med, and high) in the train and test subset.

### Extra 1: Confirmation of 75/25 split

Total train samples = 131 + 68 + 26 = 225

Train split ratio = (Total train samples)/(size of dataset) = 225/300 = 0.75

Total test samples = 44 + 22 + 9 = 75

Test split ratio = (Total test samples)/(size of dataset) = 75/300 = 0.25

Therefore, there was a proper split of 75% train set and 25% test set

### Extra 2: Confirmation of stratification

To confirm this the ratio of Train samples vs the Test samples should be approximately 3.

For Low: 131/44 = 2.977

For Med: 68/22 = 3.090

For High: 26/9 = 2.888

### Code:

# init

TRAIN\_SIZE = 0.75

labels = dataset[COLUMNS[-1]]

# perform stratified split

train\_dataset, test\_dataset = train\_test\_split(

    dataset,

    train\_size=TRAIN\_SIZE,

    stratify=labels,

    random\_state=RANDOM\_STATE

)

# Part 3: Visualization

Using the training set, for each feature, plot the feature distribution for each class. You can either use five histograms or five 1D kernel density plots. Label each sub-plot by the feature name. The distribution of feature values should be visible for all three (potentially overlapping) classes on each of the five plots. Which feature looks most useful and why? Which home efficiency class do you think will have the lowest accuracy and why? (60 words max)

## Solution:

Most useful feature: Thermal, because there is less overlap between low and high classes additionally there are a few values of high that are separate as well for larger values.

Lowest accuracy Efficiency calls: High=2, since it is overlapping with other classes in almost all the cases.

Please note that for the figures below the bin count is 30.

A graph of thermal value

Description automatically generated

Figure : Histogram of distribution for different Efficiency classes for the feature Thermal

A graph of a graph

Description automatically generated

Figure : Histogram of distribution for different Efficiency classes for the feature Area

A graph of a graph of glazing

Description automatically generated

Figure : Histogram of distribution for different Efficiency classes for the feature Glazing

A graph of a column

Description automatically generated with medium confidence

Figure : Histogram of distribution for different Efficiency classes for the feature Cladding

A graph of a roofing value

Description automatically generated

Figure : Histogram of distribution for different Efficiency classes for the feature Roofing

### Extra 1: Histogram for bin count = 10

A graph of a graph

Description automatically generated

Figure : Histogram with bin count 10 of distribution for different Efficiency classes for the feature Thermal

A graph of area value

Description automatically generated

Figure : Histogram with bin count 10 of distribution for different Efficiency classes for the feature Area

A graph of a graph

Description automatically generated with medium confidence

Figure : Histogram with bin count 10 of distribution for different Efficiency classes for the feature Glazing

A graph of a bar chart

Description automatically generated with medium confidence

Figure :Histogram with bin count 10 of distribution for different Efficiency classes for the feature Cladding

A graph of a roofing graph

Description automatically generated

Figure : Histogram with bin count 10 of distribution for different Efficiency classes for the feature Roofing

### Extra 2: Density Plots

A graph of a graph

Description automatically generated

Figure : Density plot for different efficiency classes for the feature Thermal

A graph of a number of colored lines

Description automatically generated with medium confidence

Figure : Density plot for different efficiency classes for the feature Area

A graph of a line graph

Description automatically generated

Figure : Density plot for different efficiency classes for the feature Glazing

A graph of a line graph

Description automatically generated

Figure : Density plot for different efficiency classes for the feature Cladding

A graph of a graph of a roofing value

Description automatically generated

Figure : Density plot for different efficiency classes for the feature Roofing

### Extra 3: Worst Feature

Of all the features available “Area” should be the worst performing feature.

### Code:

# init

BINS\_COUNT = 30

# build histograms for each feature

for column in COLUMNS[:-1]:

    data\_df = train\_dataset[[column,COLUMNS[-1]]]

    max\_value = data\_df[column].max()

    min\_value = data\_df[column].min()

    step = (max\_value - min\_value) / BINS\_COUNT

    range\_ = np.arange(min\_value, max\_value, step)

    for effic in range(3):

        eff\_df = data\_df[data\_df[COLUMNS[-1]] == effic]

        eff\_df[column].plot.hist(

            bins=range\_,

            histtype=u'step',

            color=COLOR[effic],

        )

        ax = eff\_df[column].plot.hist(

            bins=range\_,

            color=COLOR[effic],

            label=LABELS[effic],

            legend=True

        )

    plt.title(f"Histogram of {column}")

    plt.xlabel(f"{column} Value")

    plt.show()

# Part 4: Training with Cross Validation:

Complete 5-fold-cross-validation over the train subset using an SVM classifier with a polynomial kernel with degree=3 and C=0.8. Report the accuracy over each fold, the average accuracy across all five folds, and the standard deviation across the five accuracy measurements.

## Solution:

Fold 0: Accuracy = 0.75556

Fold 1: Accuracy = 0.82222

Fold 2: Accuracy = 0.80000

Fold 3: Accuracy = 0.77778

Fold 4: Accuracy = 0.82222

Average Accuracy: 0.7956

Standard Deviation: 0.0259

NOTE: \*\*\*\* It does not mention in the question to stratify the CV. If that is needed, please look at Extra 1.

### Extra 1: Stratified Cross Validation

Same experimentation was run again but this time with stratification in CV as well. Following were the results:

Fold 0: Accuracy = 0.75556

Fold 1: Accuracy = 0.75556

Fold 2: Accuracy = 0.73333

Fold 3: Accuracy = 0.88889

Fold 4: Accuracy = 0.84444

Average Accuracy: 0.7956

Standard Deviation: 0.0603

The mean stayed relatively the same, but the standard deviation increased.

### Extra 2: Visualization for without stratification

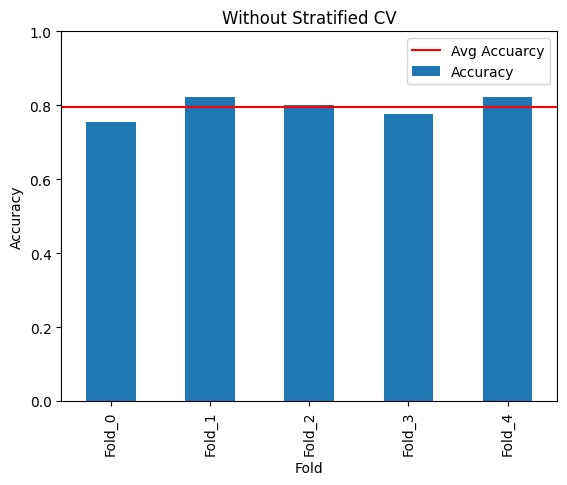


Figure : Accuracy plot for folds without stratified Cross validation.

### Extra 3: Visualization for with stratification

A graph with blue bars

Description automatically generated

Figure : Accuracy plot for folds with stratified Cross validation.

### Code

# init

kernal\_type = 'poly'

degree = 3

c = 0.8

folds = 5

# build features and lables

x = train\_dataset[COLUMNS[:-1]]

y = train\_dataset[COLUMNS[-1]]

# build SVM

svc = SVC(kernel=kernal\_type, degree=degree, C=c)

# init Cross Validation

cross\_validation = KFold(n\_splits=folds, shuffle=True, random\_state=RANDOM\_STATE)

# perform cross validation

cross\_validation\_score = cross\_val\_score(svc, x, y, cv=cross\_validation)

for i, score in enumerate(cross\_validation\_score):

    print(f'Fold {i}: Accuracy = {score:.5f}')

avg\_acc = np.mean(cross\_validation\_score)

std\_acc = np.std(cross\_validation\_score)

print(f'Average Accuracy: {avg\_acc:.4f}')

print(f'Standard Deviation: {std\_acc:.4f}')

# Part 5: Training without Cross Validation

Train another SVM model (same kernel & C) on all of your training samples. Test on the test subset. Report the accuracy on the test subset. Does it fall within 1 standard deviation of the average accuracy observed in Step 5?

## Solution

Accuracy = 0.7866

Yes, it falls under 1 standard deviation of the accuracy observed in step 4

### Extra 1: Note form the lectures

It can be seen that the accuracy drops slightly when compared with the Cross Validation set. This was a point mentioned in the lectures. This is because Cross Validation is an optimistic estimation of metrics

### Extra 2: Visualization

A graph of blue and white bars

Description automatically generated

Figure : Accuracy visualization with standard deviation

### Code:

# build SVM

svc = SVC(kernel=kernal\_type, degree=degree, C=c)

# train SVM

svc.fit(train\_dataset[COLUMNS[:-1]], train\_dataset[COLUMNS[-1]])

# calculate accuracy of SVM

svc.score(test\_dataset[COLUMNS[:-1]], test\_dataset[COLUMNS[-1]])

# Part 6: Misclassification Cost

For this question only, assume that the misclassification costs are as follows:

A table with numbers and letters

Description automatically generated

1. What is your total misclassification cost for the test set predictions from Q5 above?
2. How could you incorporate this loss information into your classifier design? (60 words)

## Solution:

1. Misclassification Cost: 23
2. Use one vs one classification to solve the multiclass problem [2]. Vary the soft margin by changing the value in the “Hinge Loss” (see extra 2) to the cost value. A pictorial diagram is give below.

A notebook with writing on it

Description automatically generated

It is to be noted that, to simply the understanding “linear” kernel is shown instead of a “polynomial”. One can easily replace the decision boundaries to polynomial.

### Extra 0: Alternative for incorporating the loss

You can simply pass in class weights in the SVC in SK-Learn. This will take into account the cost and create appropriate model.

### Extra 1: Misclassification Cost Calculation

Misclassification cost = trace(confusion\_matrix . cost\_matrixT)

But cost matrix is symmetric.

Misclassification cost = trace(confusion\_matrix . cost\_matrix)

### Extra 2: Hinge Loss Formula

Here, is always 1. is the true class, is the input feature, b is the bias term.

combined together is the score.

See reference [1] for further details.

### Extra 3: Multi class classification for SVM

SVM typically work for binary classification. Different techniques are used to expand to multi-class problem. The techniques are as follows:

1. One vs One
2. One vs All

See reference [2] for further details.

### Extra 4: References

1. <https://www.youtube.com/watch?v=IjSfa7Q8ngs>
2. <https://www.youtube.com/watch?v=3lwicUTEgHs>

### Code

cost = np.array([

    [0,1,2],

    [1,0,1],

    [2,1,0]

])

# Note y\_pred is from part 5

confusion = confusion\_matrix(test\_dataset[COLUMNS[-1]], y\_pred)

confusion

# Calculate the cost

np.trace(np.dot(confusion , cost))

# Part 7: Neural Networks (NN)

Using 5-CV across only the training subset, perform a hyperparameter sweep of the number of hidden   
nodes in a 3-layer feedforward neural network. Report your accuracy for numH=[1,10,100]hidden nodes. Use the ‘adam’ solver, a hyperbolic tangent activation function for the hidden layers.

## Solution:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| numH | Accuracy | | | | | Mean Folds Accuracy | Mean Fold  STD |
| Fold 0 | Fold 1 | Fold 2 | Fold 3 | Fold 4 |
| 1 | 0.7777 | 0.8000 | 0.7111 | 0.8666 | 0.8000 | 0.7911 | 0.0498 |
| 10 | 0.8666 | 0.9111 | 0.7111 | 0.8888 | 0.8888 | 0.8533 | 0.0724 |
| 100 | 0.8666 | 0.8888 | 0.7777 | 0.9111 | 0.8888 | 0.8666 | 0.0466 |

### Extra 1: Hyperparameters for the NN model

Epochs: 2000

Learning Rate: 0.001

Folds: 5

### Extra 2: Plots for training Losses and Accuracy for each epoch

A graph of different colored lines

Description automatically generated

Figure : Accuracy and loss for 1 hidden neuron in each fold

A graph showing different colored lines

Description automatically generated

Figure : Accuracy and loss for 10 hidden neurons in each fold

A graph of a graph

Description automatically generated

Figure : Accuracy and loss for 100 hidden neurons in each fold

### Code:

# init

num\_hidden = [1,  10,  100]

folds = 5

epochs = 2000

learning\_rate = 0.001

models = {}

losses = {}

accuracy = {}

# Define the model

class HouseEfficNN(torch.nn.Module):

    def \_\_init\_\_(self, hidden\_size):

        super(HouseEfficNN, self).\_\_init\_\_()

        self.input\_size = 5

        self.output\_size = 3

        self.hidden\_size = hidden\_size

        self.layer\_1 = torch.nn.Linear(self.input\_size, self.hidden\_size)

        self.activation = torch.nn.Tanh()

        self.layer\_2 = torch.nn.Linear(self.hidden\_size, self.output\_size)

    def forward(self, x):

        x = self.layer\_1(x)

        x = self.activation(x)

        x = self.layer\_2(x)

        return x

# Define the dataset

class HousingDataset(torch.utils.data.Dataset):

    def \_\_init\_\_(self, df, target\_column):

        self.features = torch.tensor(df.drop(columns=[target\_column]).values, dtype=torch.float32)

        self.labels = torch.tensor(df[target\_column].values, dtype=torch.long)

    def \_\_len\_\_(self):

        return len(self.labels)

    def \_\_getitem\_\_(self, idx):

        return self.features[idx], self.labels[idx]

# run training

for h\_num in num\_hidden:

    losses[h\_num] = {}

    accuracy[h\_num] = {}

    cross\_fold\_NN = StratifiedKFold(n\_splits=folds, shuffle=True, random\_state=RANDOM\_STATE)

    for fold\_index, (train\_index, test\_index) in enumerate(cross\_fold\_NN.split(train\_dataset[COLUMNS[:-1]], train\_dataset[COLUMNS[-1:]])):

        fold\_nn\_train\_df = train\_dataset.iloc[train\_index]

        fold\_nn\_train\_features, fold\_nn\_train\_labels = HousingDataset(fold\_nn\_train\_df, COLUMNS[-1])[:]

        fold\_nn\_test\_df = train\_dataset.iloc[test\_index]

        fold\_nn\_test\_features, fold\_nn\_test\_labels = HousingDataset(fold\_nn\_test\_df, COLUMNS[-1])[:]

        model = HouseEfficNN(hidden\_size=h\_num)

        optimizer = torch.optim.Adam(model.parameters(), lr=learning\_rate)

        criterion = torch.nn.CrossEntropyLoss()

        losses[h\_num][fold\_index] = []

        accuracy[h\_num][fold\_index] = []

        for epoch in tqdm(range(epochs), desc=f"H\_size: {h\_num}, Fold: {fold\_index}", unit='epoch',):

            model.train()

            optimizer.zero\_grad()

            outputs = model(fold\_nn\_train\_features)

            loss = criterion(outputs, fold\_nn\_train\_labels)

            loss.backward()

            optimizer.step()

            losses[h\_num][fold\_index].append(loss.item())

            model.eval()

            predictions = model(fold\_nn\_test\_features)

            acc = accuracy\_score(fold\_nn\_test\_labels, predictions.argmax(1).detach().numpy())

            accuracy[h\_num][fold\_index].append(acc)

        models[h\_num] = model

# Part 8: Naïve Bayes Classifier

Returning to Question 3, compare a naïve Bayes classifier trained using only the ‘most useful’ feature to a naïve Bayes classifier trained using all five features. Describe how you split/used your data, how you tested the hypothesis (null hypothesis, alternative hypothesis, test metric, etc.), what p-value you obtained, and your conclusion.

## Solution:

**H0:** The accuracy values for 5 feature classifier is similar to the accuracy of the 1 feature (Thermal) classifier

**H1:** The accuracy values for 5 feature classifier is different from the accuracy values of 1 feature (Thermal) classifier

**Data splitting**: The data was first split into test (25%) and train (75%). On these 2 sets of data bootstrapping was conducted with replacements. On each iteration the test data was randomly sampled form the test set with repetition and the train data was randomly sampled with repetition from the train data. In each iteration the total number of data points (replacement samples) in the train set was 75% of 300, and total number of data points (replacement samples) for test set was 25% of 300.

**Test metrics:** accuracy

Accuracy = total number of correctly classified samples / total number of samples

**Hypothesis Testing:**

2 sample Bootstrapping was used to perform hypothesis testing. The repeated bootstrapped samples were taken as described in “Data splitting” and accuracy was measured for classifier with all 5 features and 1 feature (thermal) for each iteration. A further description of the process is found in “SYSC5405-Slides-04-ClassificationAccuracy” from slides 60 to 67. The code also a provided good understanding of the test. It is provided below.

**P-value:** 0.49924

**Conclusion:**

Given the p-values is above the significance level of 0.05, we cannot reject the null hypothesis.

In other works we fail to reject that there is a significance difference in the accuracy of the two classifiers.

### Extra 1: Significance Level (alpha)

For this problem the significance level was 0.05.

The confidence level is 95%

### Extra 2: Number of iterations

For this problem 9,999 iterations were conducted.

### Extra 3: Distribution of mean under H0

A graph of a number of objects

Description automatically generated

Figure : Distribution of Mean under H0

### Code

# init

num\_iters = 9999

accuracies\_5\_features = []

accuracies\_1\_features = []

accuracies\_diff = []

# calculate the diffrence of accuracies for the two models

real\_classifier\_5 = GaussianNB()

real\_classifier\_5.fit(train\_dataset[COLUMNS[:-1]], train\_dataset[COLUMNS[-1]])

y\_pred\_real\_5 = real\_classifier\_5.predict(test\_dataset[COLUMNS[:-1]])

accuracy\_5 = accuracy\_score(test\_dataset[COLUMNS[-1]], y\_pred\_real\_5)

real\_classifier\_1 = GaussianNB()

real\_classifier\_1.fit(train\_dataset[[COLUMNS[0]]], train\_dataset[COLUMNS[-1]])

y\_pred\_real\_1 = real\_classifier\_1.predict(test\_dataset[[COLUMNS[0]]])

accuracy\_1 = accuracy\_score(test\_dataset[COLUMNS[-1]], y\_pred\_real\_1)

real\_diff = accuracy\_1 - accuracy\_5

print(f"1 feature: {accuracy\_1}")

print(f"5 feature: {accuracy\_5}")

gnb = GaussianNB()

for \_ in range(num\_iters):

    iter\_train\_df = train\_dataset.sample(

        len(train\_dataset),

        replace = True

    )

    iter\_test\_df = test\_dataset.sample(

        len(test\_dataset),

        replace = True

    )

    gnb.fit(iter\_train\_df[COLUMNS[:-1]], iter\_train\_df[COLUMNS[-1]])

    y\_pred\_5 = gnb.predict(iter\_test\_df[COLUMNS[:-1]])

    accuracy\_5 = accuracy\_score(iter\_test\_df[COLUMNS[-1]], y\_pred\_5)

    accuracies\_5\_features.append(accuracy\_5)

    # nb\_classifier\_1 = GaussianNB()

    gnb.fit(iter\_train\_df[[COLUMNS[0]]], iter\_train\_df[COLUMNS[-1]])

    y\_pred\_1 = gnb.predict(iter\_test\_df[[COLUMNS[0]]])

    accuracy\_1 = accuracy\_score(iter\_test\_df[COLUMNS[-1]], y\_pred\_1)

    accuracies\_1\_features.append(accuracy\_1)

    diff\_accuracy = accuracy\_1 - accuracy\_5

    accuracies\_diff.append(diff\_accuracy)

p\_val = sum([diff >= real\_diff for diff in accuracies\_diff])/num\_iters

print(f”p-value {p\_val}”)

# Appendix:

## Unimportant code:

import torch

import torch

import numpy as np

import pandas as pd

from tqdm import tqdm

from sklearn.svm import SVC

import matplotlib.pyplot as plt

from torch.utils.data import Dataset, DataLoader

from sklearn.model\_selection import train\_test\_split, cross\_val\_score, KFold, StratifiedKFold

from sklearn.metrics import classification\_report, accuracy\_score, confusion\_matrix

from sklearn.naive\_bayes import GaussianNB

device = torch.device('cpu')

RANDOM\_STATE = 1

ALPHA = 0.5

COLOR = {

    0: (1.0,0.0,0.0,ALPHA),

    1: (1.0,1.0,0.0,ALPHA),

    2: (0.0,1.0,0.0,ALPHA)

}

LABELS = {

    0: 'low',

    1: 'med',

    2: 'high'

}