Homework 6

Due: **November 5th, 5pm** (late submission until November 8th, 5pm -- no submission possible afterwards)

Written assignment: 15 points

Coding assignment: 20 points

Project report: 10 points

Name: [Yawen Tan]

Link to the github repo: [https://github.com/IsabellaTan/Brown-DATA2060-HW6]

Written Assignment (15 points)

Problem 1: VC Dimension

For any hypothesis space $\mathcal{H}$ on domain $\mathcal{X}$, to show that the VC Dimension of $\mathcal{H}$ is d, you should prove each of the following:

- There exists a set $C \subset \mathcal{X}$ of size d such that $\mathcal{H}$ shatters C
- There does not exist a set $C' \subset \mathcal{X}$ of size $d+1$ such that $\mathcal{H}$ shatters C'

Part 1

Compute and prove the VC dimension for the following hypothesis spaces:

a. The class of signed intervals in $\mathbb{R}$, $\mathcal{H} = \{h_{a,b,s} : a \leq b, s \in \{-1, 1\}\}$ where:

$$h_{a,b,s} = \begin{cases} s & \text{if } x \in [a,b] \\ -s & \text{if } x \notin [a,b] \end{cases}$$

b. The class of origin-centered spheres in $\mathbb{R}^d$, $\mathcal{H} = \{h_{a,s} : s \in \{-1, 1\}, a \in \mathbb{R}\}$ where:

$$h_{a,s} = \begin{cases} s & \text{if } x \text{ is within or on the origin centered sphere of radius } a \\ -s & \text{if } x \text{ is outside the origin centered sphere of radius } a \end{cases}$$

Solutions:

a.

VC dimension = 3

We firstly prove that there exists a set $C \subset \mathcal{X}$ of size 3 such that h shatters C

Let x_1 < x_2 < x_3, m=(x_1 + x_2) / 2, n= m=(x_2 + x_3) / 2

We will show that for every possible labeling of the three points x_1,x_2,x_3 (2^3=8 total), we can find an interval [a,b] and a sign s∈{−1,1} such that the classifier matches that labeling.

- [1, 1, 1]: let s=1, [a,b] = [x_1, x_3]
- [-1, 1, 1]: let s=1, [a,b] = [x_2, x_3]
- [1, -1, 1]: let s=-1, [a,b] = [m, n]
- [1, 1, -1]: let let s=1, [a,b] = [x_1, x_2]
- [-1, -1, 1]: let s=-1, [a,b] = [x_1, x_2]
- [-1, 1, -1]: let s=1, [a,b] = [m, n]
- [1, -1, -1]: let s=-1, [a,b] = [x_2, x_3]
- [-1, -1, -1]: let s=-1, [a,b] = [x_1, x_3]

Then we prove that there does not exist a set $C' \subset \mathcal{X}$ of size 4 such that h shatters C'

Let x_1 < x_2 < x_3 < x_4

Consider the labeling pattern (1,−1,1,−1) which is in the set of size 4, we cannot shatter.

When s=1, if [a,b] = [x_1, x_3], for x_1, x_3, it will have value '1', but '−1' at x_2 . A single interval can only cover one continuous block of points, so it can't include both x_1 and x_3 without also including x_2.

When s=-1, if [a,b] = [x_2, x_4], for x_2, x_4, it will have value '-1', but '1' at x_3. $A single interval can only cover one continuous block of points, so it can't include both x\_2 and x\_4 without also including$

x_3$.

Therefore, VC dimension = 3.

b.

VC dimension = 2

We firstly prove that there exists a set $C \subset \mathcal{X}$ of size 2 such that h shatters C

Let x_1, x_2 in $\mathbb{R}^d$ with $\|x_1\| < \|x_2\|$

All four labelings of (x_1, x_2) are realizable:

- [1,1]: take s=1, a≥$\|x_2\|$ (both inside).
- [1,−1]: take s=1, a ∈ [$\|x_1\|$, $\|x_2\|$) (only x_1 inside).
- [−1,1]: take s=−1, a∈[$\|x_1\|$, $\|x_2\|$) (inside is −1, outside is +1).
- [−1,−1]: take s=−1, a≥$\|x_2\|$ (both inside, hence −1).

Then we prove that there does not exist a set $C' \subset \mathcal{X}$ of size 3 such that h shatters C'

Let x_1, x_2, x_3 be any three points ordered by radius $\|x1\| < \|x2\| < \|x3\|$.

The classifier still depends on only one sphere. The issue is that if we tune a to correctly realize one pattern like [1,−1,1], then with the same set of points it will be impossible to realize other labelings like [−1,−1,1]. The shape is symmetric and continuous, so it can only separate the points once—it cannot alternate between positive and negative labels multiple times. As a result, although some of the 8 possible patterns can be achieved, not all 8 label combinations can hold for the same three points. Therefore, three points cannot be shattered.

Therefore, VC dimension = 2.

Part 2

Consider two hypothesis spaces $\mathcal{H}_1$, $\mathcal{H}_2$ such that $\mathcal{H}_1 \subset \mathcal{H}_2$. Prove that the VC Dimension of $\mathcal{H}_2$ is at least as large as the VC Dimension of $\mathcal{H}_1$.

Solution:

Let d = VCdim(H1), then there exit a set C={x1, x2, ...xn} ⊆ X such that H1 shatter C

That is, for every possible labeling l: C->{0,1}, there exists some h_l such that: $h_l(x_i) = l(x_i)$ for all $x_i \in$ C.

Since $\mathcal{H}_1$, $\mathcal{H}_2$, we also have $h_l \in H_2$

Therefore, H_2 can realize the same labelings on C.

Hence, H_2 also shatters C

It follows that H_2 can shatter at least one set of size d, so VCdim(H_2) >= d = VCdim(H_1)

Programming Assignment (20 points)

Introduction

In this assignment, you'll implement Naive Bayes and use this algorithm to classify the credit rating (good or bad) of a set of individuals. The textbook section relevant to this assignment is 24.2 on page 347.

Stencil Code & Data

We have provided the following stencils:

- `Models` contains the `NaiveBayes` model which you will be implementing.
- `Check Model` contains a series of tests to ensure you are coding your model properly.
- `Main` is the entry point of your program which will read in the data, run the classifiers and print the results. Note that pre-processing has been done for you; feel free to examine the code for what exactly was done.

You should *not* modify any code in the `Main`. All the functions you need to fill in reside in `Models`, marked by `TODO`s. You can see a full description of them in the section below.

German Credit Dataset

You will be using the commonly-used German Credit dataset, which includes 1000 total examples. The prediction task is to decide whether someone's credit is good (1) or bad (0). A full list of attributes can be found **here**; note that this includes sensitive attributes like sex, age, and personal status. The specific file we are using comes from **Friedler et.al., 2019**. This data is in the file `german_numerical-binsensitive.csv`.

Data Format

The original feature values in this dataset are mixed---some categorical, some numerical. We have written all the preprocessing code for you, transforming numerical attributes into categories and encoding all attributes as binary features. After preprocessing, there are a total of 69 attributes which take on either 1 or 0. **`credit = 1` corresponds to "good" credit, and `credit = 0` corresponds to "bad" credit.**

The Assignment

In `Models`, there are three functions you will implement. They are:

- `NaiveBayes`:
  - **train()** uses maximum likelihood estimation to learn the parameters (attribute distributions and priors distribution). Because all the features are binary values, you should use the Bernoulli distribution (as described in lecture) for the features. Remember to add Laplace smoothing as you calculate the distributions.
  - **predict()** predicts the labels using the inputs of test data. You should return 1-D numpy array.
  - **accuracy()** computes the percentage of the correctly predicted labels over a dataset.

Note that there is also a **print_fairness()** method implemented for you in `NaiveBayes`. You should not change this method. Additionally, you are not allowed to use any off-the-shelf packages that have already implemented Naive Bayes, such as scikit-learn; we're asking you to implement it yourself.

In [1]:
```
from __future__ import print_function
from packaging.version import parse as Version
from platform import python_version

OK = '\x1b[42m[ OK ]\x1b[0m'
FAIL = "\x1b[41m[FAIL]\x1b[0m"

try:
    import importlib
except ImportError:
    print(FAIL, "Python version 3.12.11 is required,"
                " but %s is installed." % sys.version)

def import_version(pkg, min_ver, fail_msg=""):
    mod = None
    try:
        mod = importlib.import_module(pkg)
        if pkg in {'PIL'}:
            ver = mod.VERSION
        else:
            ver = mod.__version__
        if Version(ver) == Version(min_ver):
            print(OK, "%s version %s is installed."
                  % (lib, min_ver))
        else:
            print(FAIL, "%s version %s is required, but %s installed."
                  % (lib, min_ver, ver))
    except ImportError:
        print(FAIL, '%s not installed. %s' % (pkg, fail_msg))
    return mod


# first check the python version
pyversion = Version(python_version())

if pyversion >= Version("3.12.11"):
    print(OK, "Python version is %s" % pyversion)
elif pyversion < Version("3.12.11"):
    print(FAIL, "Python version 3.12.11 is required,"
                " but %s is installed." % pyversion)
else:
    print(FAIL, "Unknown Python version: %s" % pyversion)


print()
requirements = {'matplotlib': "3.10.5", 'numpy': "2.3.2",'sklearn': "1.7.1",
                'pandas': "2.3.2", 'pytest': "8.4.1", 'torch':"2.7.1"}

# now the dependencies
```

```
for lib, required_version in list(requirements.items()):
    import_version(lib, required_version)
```

```
[ OK ] Python version is 3.12.11

[ OK ] matplotlib version 3.10.5 is installed.
[ OK ] numpy version 2.3.2 is installed.
[ OK ] sklearn version 1.7.1 is installed.
[ OK ] pandas version 2.3.2 is installed.
[ OK ] pytest version 8.4.1 is installed.
[ OK ] torch version 2.7.1 is installed.
```

Model

In [5]:

```
import numpy as np
import pandas as pd

class NaiveBayes(object):
    """ Bernoulli Naive Bayes model

    @attrs:
        n_classes:    the number of classes
        attr_dist:    a 2D (n_classes x n_attributes) NumPy array of the attribute distributions
        label_priors: a 1D NumPy array of the priors distribution
    """

    def __init__(self, n_classes):
        """ Initializes a NaiveBayes model with n_classes. """
        self.n_classes = n_classes
        self.attr_dist = None
        self.label_priors = None

    def train(self, X_train, y_train):
        """ Trains the model, using maximum likelihood estimation.
        @params:
            X_train: a 2D (n_examples x n_attributes) numpy array
            y_train: a 1D (n_examples) numpy array
        @return:
            a tuple consisting of:
                1) a 2D numpy array of the attribute distributions
                2) a 1D numpy array of the priors distribution
        """

        # TODO
        # n_examples = number of samples (rows)
        # n_attributes = number of binary features (columns)
        n_examples, n_attributes = X_train.shape
        K = self.n_classes # K = number of possible classes
        alpha = 1.0 # Laplace smoothing factor.

        # Count how many samples belong to each class.
        counts_per_class = np.array([(y_train == c).sum() for c in range(K)], dtype=float)
        # Compute the prior probability for each class:
        # P(Y=c) = (N_c + α) / (N + α*K)
        label_priors = (counts_per_class + alpha) / (n_examples + alpha * K)
        # Initialize a 2D array to hold conditional probabilities for each feature:
        # attr_dist[c, j] = P(X_j = 1 | Y = c)
        attr_dist = np.zeros((K, n_attributes), dtype=float)

        # Loop over each class and estimate its feature probabilities
        for c in range(K):
            # Create a boolean mask selecting all samples that belong to class c
            idx = (y_train == c)
            Nc = int(idx.sum()) # number of sample of this class
            if Nc > 0:
                # Sum across rows (samples) within this class to count how often each feature = 1
                sum_ones = X_train[idx].sum(axis=0)
            else:
                # If there are no samples of this class in the training set, just use a vector of zeros
                sum_ones = np.zeros(n_attributes, dtype=float)
            # Compute Bernoulli conditional probabilities with Laplace smoothing:
            # P(X_j=1 | Y=c) = (sum_ones_j + α) / (N_c + 2α)
            attr_dist[c, :] = (sum_ones + alpha) / (Nc + 2.0 * alpha)


        self.attr_dist = attr_dist
        self.label_priors = label_priors
        return self.attr_dist, self.label_priors

    def predict(self, inputs):
        """ Outputs a predicted label for each input in inputs.
            Remember to convert to log space to avoid overflow/underflow
            errors!

        @params:
            inputs: a 2D NumPy array containing inputs
```

```
        @return:
            a 1D numpy array of predictions
        """

        # TODO
        X = np.asarray(inputs)
        # Take the log of learned parameters.
        log_theta = np.log(self.attr_dist) # log P(X_j=1 | Y=c)
        log_one_minus = np.log(1.0 - self.attr_dist) # log P(X_j=0 | Y=c)
        log_priors = np.log(self.label_priors) # log P(Y=c)

        n_samples, d = X.shape # number of samples and features
        K = self.attr_dist.shape[0] # number of classes
        scores = np.empty((n_samples, K), dtype=float)
        # For each class, compute the total log-probability of every sample
        for c in range(K):
            scores[:, c] = (
                log_priors[c]
                + X @ log_theta[c, :].T
                + (1.0 - X) @ log_one_minus[c, :].T
            )
        preds = np.argmax(scores, axis=1).astype(int)
        return preds

    def accuracy(self, X_test, y_test):
        """ Outputs the accuracy of the trained model on a given dataset (data).

        @params:
            X_test: a 2D numpy array of examples
            y_test: a 1D numpy array of labels
        @return:
            a float number indicating accuracy (between 0 and 1)
        """

        # TODO
        y_test = np.asarray(y_test).astype(int)
        preds = self.predict(X_test)
        return np.mean(preds == y_test)

    def print_fairness(self, X_test, y_test, x_sens):
        """
        ***DO NOT CHANGE what we have implemented here.***

        Prints measures of the trained model's fairness on a given dataset (data).

        For all of these measures, x_sens == 1 corresponds to the "privileged"
        class, and x_sens == 0 corresponds to the "disadvantaged" class. Remember that
        y == 1 corresponds to "good" credit.

        @params:
            X_test: a 2D numpy array of examples
            y_test: a 1D numpy array of labels
            x_sens: a numpy array of sensitive attribute values
        @return:

        """
        predictions = self.predict(X_test)

        # Disparate Impact (80% rule): A measure based on base rates: one of
        # two tests used in legal literature. All unprivileged classes are
        # grouped together as values of 0 and all privileged classes are given
        # the class 1. . Given data set D = (S,X,Y), with protected
        # attribute S (e.g., race, sex, religion, etc.), remaining attributes X,
        # and binary class to be predicted Y (e.g., “will hire”), we will say
        # that D has disparate impact if:
        # P[Y^ = 1 | S != 1] / P[Y^ = 1 | S = 1] <= (t = 0.8).
        # Note that this 80% rule is based on US legal precedent; mathematically,
        # perfect "equality" would mean

        di = np.mean(predictions[np.where(x_sens==0)])/np.mean(predictions[np.where(x_sens==1)])
        print("Disparate impact: " + str(di))

        # Group-conditioned error rates! False positives/negatives conditioned on group

        pred_priv = predictions[np.where(x_sens==1)]
        pred_unpr = predictions[np.where(x_sens==0)]
        y_priv = y_test[np.where(x_sens==1)]
        y_unpr = y_test[np.where(x_sens==0)]

        # s-TPR (true positive rate) = P[Y^=1|Y=1,S=s]
        priv_tpr = np.sum(np.logical_and(pred_priv == 1, y_priv == 1))/np.sum(y_priv)
        unpr_tpr = np.sum(np.logical_and(pred_unpr == 1, y_unpr == 1))/np.sum(y_unpr)

        # s-TNR (true negative rate) = P[Y^=0|Y=0,S=s]
        priv_tnr = np.sum(np.logical_and(pred_priv == 0, y_priv == 0))/(len(y_priv) - np.sum(y_priv))
        unpr_tnr = np.sum(np.logical_and(pred_unpr == 0, y_unpr == 0))/(len(y_unpr) - np.sum(y_unpr))
```

```
    # s-FPR (false positive rate) = P[Y^=1|Y=0,S=s]
    priv_fpr = 1 - priv_tnr
    unpr_fpr = 1 - unpr_tnr

    # s-FNR (false negative rate) = P[Y^=0|Y=1,S=s]
    priv_fnr = 1 - priv_tpr
    unpr_fnr = 1 - unpr_tpr

    print("FPR (priv, unpriv): " + str(priv_fpr) + ", " + str(unpr_fpr))
    print("FNR (priv, unpriv): " + str(priv_fnr) + ", " + str(unpr_fnr))


    # #### ADDITIONAL MEASURES IF YOU'RE CURIOUS #####

    # Calders and Verwer (CV) : Similar comparison as disparate impact, but
    # considers difference instead of ratio. Historically, this measure is
    # used in the UK to evalutate for gender discrimination. Uses a similar
    # binary grouping strategy. Requiring CV = 1 is also called demographic
    # parity.

    cv = 1 - (np.mean(predictions[np.where(x_sens==1)]) - np.mean(predictions[np.where(x_sens==0)]))

    # Group Conditioned Accuracy: s-Accuracy = P[Y^=y|Y=y,S=s]

    priv_accuracy = np.mean(predictions[np.where(x_sens==1)] == y_test[np.where(x_sens==1)])
    unpriv_accuracy = np.mean(predictions[np.where(x_sens==0)] == y_test[np.where(x_sens==0)])

    return predictions
```

Check Model

In [ ]:

```
# DO NOT EDIT!

import pytest
# Sets random seed for testing purposes
np.random.seed(0)

# Creates Test Models with 2 & 3 classes
test_model1 = NaiveBayes(2)
test_model2 = NaiveBayes(2)
test_model3 = NaiveBayes(3)

# Creates Test Data
x1 = np.array([[0,0,1], [0,1,0], [1,0,1], [1,1,1], [0,0,1]])
y1 = np.array([0,0,1,1,0])
x_test1 = np.array([[1,0,0],[0,0,0],[1,1,1],[0,1,0], [1,1,0]])
y_test1 = np.array([0,0,1,0,1])

x2 = np.array([[0,0,1], [0,1,1], [1,1,1], [1,1,1], [0,0,0], [1,1,0]])
y2 = np.array([0,1,1,1,0,1])
x_test2 = np.array([[0,0,1], [0,1,1], [1,1,1], [1,0,0]])
y_test2 = np.array([0,1,1,0])

x3 = np.array([[0,0,0,0], [0,0,0,1], [0,0,1,0], [0,0,1,1], [0,1,0,0], [0,1,0,1],
               [0,1,1,0], [0,1,1,1], [1,0,0,0], [1,0,0,1], [1,0,1,0], [1,0,1,1]])
y3 = np.array([0, 0, 1, 1, 1, 0, 2, 0, 0, 2, 1, 1])

x_test3 = np.array([[1, 1, 0, 0], [1, 1, 0, 1], [1, 1, 1, 0], [1, 1, 1, 1]])
y_test3 = np.array([1, 1, 0, 0])

# Test Models
def check_train_dtype(model, attrs, priors, x_train, y_train):
    assert isinstance(attrs, np.ndarray)
    assert attrs.ndim==2 and attrs.shape==(model.n_classes, x_train.shape[1])
    assert isinstance(priors, np.ndarray)
    assert priors.ndim==1 and priors.shape==(model.n_classes, )


attrs1, priors1 = test_model1.train(x1,y1)
check_train_dtype(test_model1, attrs1, priors1, x1, y1)
assert (attrs1 == pytest.approx(np.array([[.2, .4, .6],[.75, .5, .75]]),0.01))
assert (priors1 == pytest.approx(np.array([0.571, 0.429]), 0.01))

attrs2, priors2 = test_model2.train(x2, y2)
check_train_dtype(test_model2, attrs2, priors2, x2, y2)
assert (attrs2 ==  pytest.approx(np.array([[.25, .25, .5],[.67, .83, .67]]), 0.01))
assert (priors2 == pytest.approx(np.array([0.375, 0.625]), 0.01))

attrs3, priors3 = test_model3.train(x3, y3)
check_train_dtype(test_model3, attrs3, priors3, x3, y3)
assert (attrs3 == pytest.approx(np.array([[0.28571,0.428571,0.28571,0.57142],[0.42857,0.28571,0.71428,0.428571],[0.5,0.5,0.5,0
assert (priors3 == pytest.approx(np.array([0.4, 0.4, 0.2]), 0.01))

# Test Model Predictions
def check_test_dtype(pred, x_test):
    assert isinstance(pred,np.ndarray)
```

```
    assert pred.ndim==1 and pred.shape==(x_test.shape[0], )

pred1 = test_model1.predict(x_test1)
check_test_dtype(pred1, x_test1)
assert (pred1 == np.array([1, 0, 1, 0, 1])).all()

pred2 = test_model2.predict(x_test2)
check_test_dtype(pred2, x_test2)
assert (pred2 == np.array([0, 1, 1, 0])).all()

pred3 = test_model3.predict(x_test3)
check_test_dtype(pred3, x_test3)
assert (pred3 == np.array([0, 0, 1, 1])).all()

# Test Model Accuracy
assert test_model1.accuracy(x_test1, y_test1) == .8
assert test_model2.accuracy(x_test2, y_test2) == 1.0
assert test_model3.accuracy(x_test3, y_test3) == 0.0
```

Main

In [8]:

```
def get_credit():
    """
    Gets and preprocesses German Credit data
    """
    data = pd.read_csv('./data/german_numerical-binsensitive.csv') # Reads file - may change

    # MONTH categorizing
    data['month'] = pd.cut(data['month'],3, labels=['month_1', 'month_2', 'month_3'], retbins=True)[0]
    # month bins: [ 3.932     , 26.66666667, 49.33333333, 72.        ]
    a = pd.get_dummies(data['month'])
    data = pd.concat([data, a], axis = 1)
    data = data.drop(['month'], axis=1)

    # CREDIT categorizing
    data['credit_amount'] = pd.cut(data['credit_amount'], 3, labels=['cred_amt_1', 'cred_amt_2', 'cred_amt_3'], retbins=True)[
    # credit bins: [  231.826,  6308.   , 12366.   , 18424.   ]
    a = pd.get_dummies(data['credit_amount'])
    data = pd.concat([data, a], axis = 1)
    data = data.drop(['credit_amount'], axis=1)

    for header in ['investment_as_income_percentage', 'residence_since', 'number_of_credits']:
        a = pd.get_dummies(data[header], prefix=header)
        data = pd.concat([data, a], axis = 1)
        data = data.drop([header], axis=1)

    # change from 1-2 classes to 0-1 classes
    data['people_liable_for'] = data['people_liable_for'] -1
    data['credit'] = -1*(data['credit']) + 2 # original encoding 1: good, 2: bad. we switch to 1: good, 0: bad

    # balance dataset
    data = data.reindex(np.random.permutation(data.index)) # shuffle
    pos = data.loc[data['credit'] == 1]
    neg = data.loc[data['credit'] == 0][:350]
    combined = pd.concat([pos, neg])

    y = data.iloc[:, data.columns == 'credit'].to_numpy()
    x = data.drop(['credit', 'sex', 'age', 'sex-age'], axis=1).to_numpy()

    # split into train and validation
    X_train, X_val, y_train, y_val = x[:350, :], x[351:526, :], y[:350, :].reshape([350,]), y[351:526, :].reshape([175,])

    # keep info about sex and age of validation rows for fairness portion
    x_sex = data.iloc[:, data.columns == 'sex'].to_numpy()[351:526].reshape([175,])
    x_age = data.iloc[:, data.columns == 'age'].to_numpy()[351:526].reshape([175,])
    x_sex_age = data.iloc[:, data.columns == 'sex-age'].to_numpy()[351:526].reshape([175,])

    return X_train, X_val, y_train, y_val, x_sex, x_age, x_sex_age


np.random.seed(0)
X_train, X_val, y_train, y_val, x_sex, x_age, x_sex_age = get_credit()
model = NaiveBayes(2)
model.train(X_train, y_train)

print("------------------------------------------------------------")
print("Train accuracy:")
print(model.accuracy(X_train, y_train))
print("------------------------------------------------------------")
print("Test accuracy:")
print(model.accuracy(X_val, y_val))
print("------------------------------------------------------------")

print("Fairness measures:")
model.print_fairness(X_val, y_val, x_sex_age)
```

```
------------------------------------------------------------
Train accuracy:
0.7742857142857142
------------------------------------------------------------
Test accuracy:
0.7257142857142858
------------------------------------------------------------
Fairness measures:
Disparate impact: 0.8294586797895808
FPR (priv, unpriv): 0.7083333333333333, 0.37037037037037035
FNR (priv, unpriv): 0.17500000000000004, 0.15909090909090906
```

Out[8]:
```
array([1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0,
       0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0,
       0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0,
       1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0])
```

Project Report (10 points)

Question 1

Report the training and testing accuracy of the Naive Bayes classifier. (A correct implementation should have testing accuracy above 70%.)

Solution:

The model was trained on the preprocessed German Credit dataset, where all attributes were converted into binary indicators.Laplace smoothing was applied when estimating both the feature likelihoods and class priors to avoid zero probabilities. From the program output, we can get training accuracy is 0.774 (77.4 %), testing accuracy is 0.726 (72.6 %). These results show that the Naive Bayes model fits the training data reasonably well with over 70 % accuracy.

Question 2

What strong assumption about the features/attributes of the data does Naive Bayes make? Comment on this assumption in the context of credit scores.

Solution:

Naive Bayes relies on the idea that once the credit label is known, every feature behaves independently of the others. In simple terms, the model treats information like income, loan amount, and job type as if they were unrelated when predicting whether a person has good or bad credit.

In real financial data, this is rarely true — people with high income usually have stronger credit histories and tend to borrow more responsibly. Even so, the model often performs reasonably well because it captures the main statistical tendencies rather than the exact relationships between variables.

Question 3

This dataset was originally structured as follows:

Month	Credit Amount	Number of credits	...	Credit
6	1169	2	...	1
48	5951	1	...	2
12	2096	1	...	1
9	2134	3	...	1

For each of the above attributes, describe what transformations to the original dataset would need to occur for it to be usable in a Bernoulli Naive Bayes model. *(hint: every attribute must take on the value of 0 or 1)*

Solution:

Bernoulli Naive Bayes requires every feature to be binary (0 or 1). Therefore, each original numeric or categorical column must be converted into one-hot or binned binary indicators. In general, categorical variables and continuous features with a small number of distinct values can be directly one-hot encoded, while continuous variables with a wide numeric range should first be grouped into several ranges (bins) before encoding.

For number of credits and credit, we can use one-hot encoding to convert the value into value of 0 or 1. Because the possible values are limited not widely spread, then one-hot encoding is sufficient. For example: if the variable 'credits' takes values {1, 2, 3}, we can create three new binary columns — credits_1, credits_2, and credits_3 — where only one of them equals 1 for each record, and other equal 0.

For month and credit amount, we can divide each variable into different intervals, after separating, we use one-hot encoding. Because the possible values widely spread, if we use one-hot encoding directly, it might have too many columns and the model may not learn well. For example: if credit_amount ranges from 100 to 10,000, we can group it into three parts — low, medium, and high — and then create three binary columns: cred_amt_low, cred_amt_medium, and cred_amt_high. Each record will have exactly one of these equal to 1, and other equal 0.

Question 4

A different way to think about fairness is based on the errors the model makes. We define the false positive rate (FPR) as $P(\hat{Y} = 1|Y = 0)$, and the false negative rate (FNR) as $P(\hat{Y} = 0|Y = 1)$. Suppose we calculate FPR and FNR for each group. In words, what does the false positive rate and false negative rate represent in the context of credit ratings? What are the implications if one group's FPR is much higher than the other's? What are the implications if one group's FNR is much higher than the other's?

Solution:

False Positive Rate (FPR) means among people who actually have bad credit, how many are incorrectly predicted as good credit. A high FPR means the model gives loans to risky borrowers, which increases financial loss risk.

False Negative Rate (FNR) means among people who actually have good credit, how many are incorrectly predicted as bad credit. A high FNR means many creditworthy individuals are unfairly denied loans.

If one group has a much higher FPR, the bank takes greater financial risk on that group. If one group has a much higher FNR, that group faces systematic discrimination, being wrongly rejected more often. Fair credit models should strive for balanced FPR and FNR across demographic groups.