Aprendizagem 2021/22

Homework I – Group 039

I. Pen-and-paper

1)

Priors:

$$\left.\begin{aligned} P(C=0) &= \frac{4}{10} \\ P(C=1) &= \frac{6}{10} \end{aligned}\right\} Priors$$

Y1:

$\mu(C=0) = 0.25$
$\mu(C=1) = 0.05$
$\sigma(C=0) = 0.24$
$\sigma(C=1) = 0.29$

Y2:

$P(Y_2 = A|C=0) = \frac{2}{4}$

$P(Y_2 = B|C=0) = \frac{1}{4}$

$P(Y_2 = C|C=0) = \frac{1}{4}$

$P(Y_2 = A|C=1) = \frac{1}{6}$

$P(Y_2 = B|C=1) = \frac{2}{6}$

$P(Y_2 = C|C=1) = \frac{3}{6}$

Y3 & Y4:

$\mu(C=0) = \begin{bmatrix} 0.20 \\ 0.25 \end{bmatrix}$

$\mu(C=1) = \begin{bmatrix} 0.12 \\ 0.08 \end{bmatrix}$

$\Sigma(C=0) = \begin{bmatrix} 0.18 & 0.18 \\ 0.18 & 0.25 \end{bmatrix}$

$\Sigma(C=1) = \begin{bmatrix} 0.11 & 0.12 \\ 0.12 & 0.21 \end{bmatrix}$

$$P(Class = 0\ |Y_1 = y_1, Y_2 = y_2, Y_3 = y_3, Y_4 = y_4) =$$

$$= \frac{P(Y_1 = y_1|class = 0) \times P(Y_2 = y_2|class = 0) \times P(Y_3 = y_3, Y_4 = y_4|class = 0) \times P(Class = 0)}{P(Y_1 = y_1) \times P(Y_2 = y_2) \times P(Y_3 = y_3, Y_4 = y_4)}$$

$$= \frac{N(y_1|0.25, 0.24^2) \times P(Y_2 = y_2|Class = 0) \times N\left(\begin{bmatrix} y_3 \\ y_4 \end{bmatrix} \middle| \begin{bmatrix} 0.20 \\ 0.25 \end{bmatrix}, \begin{bmatrix} 0.42 & 0.18 \\ 0.18 & 0.50 \end{bmatrix}\right) \times \frac{4}{10}}{P(Y_1 = y_1) \times P(Y_2 = y_2) \times P(Y_3 = y_3, Y_4 = y_4)}$$

$$P(Class = 1\,|Y_1 = y_1, Y_2 = y_{2,}Y_3 = y_{3,}Y_4 = y_4) =$$

$$= \frac{P(Y_1 = y_1|class = 1) \times P(Y_2 = y_2|class = 1) \times P(Y_3 = y_3, Y_4 = y_4|class = 1) \times P(Class = 1)}{P(Y_1 = y_1) \times P(Y_2 = y_2) \times P(Y_3 = y_{3,}Y_4 = y_4)}$$

$$= \frac{N(y_1|0.05, 0.2881^2) \times P(Y_2 = y_2|Class = 1) \times N\left(\begin{bmatrix} y_3 \\ y_4 \end{bmatrix} \middle| \begin{bmatrix} 0.12 \\ 0.08 \end{bmatrix}, \begin{bmatrix} 0.33 & 0.12 \\ 0.12 & 0.46 \end{bmatrix}\right) \times \frac{6}{10}}{P(Y_1 = y_1) \times P(Y_2 = y_2) \times P(Y_3 = y_{3,}Y_4 = y_4)}$$

2)

Appling the trained model to the data set

	P(Class = 0)	P(Class = 1)
x_1	0.84	0.16
x_2	0.20	0.80
x_3	0.76	0.24
x_4	0.46	0.54
x_5	0.46	0.54
x_6	0.07	0.93
x_7	0.06	0.94
x_8	0.47	0.53
x_9	0.70	0.30
x_{10}	0.09	0.91

Predicted|True	Positive	Negative	*Total*
Positive	5	2	*7*
Negative	1	2	*3*
Total	6	4	*10*

3)

$$accuracy = \frac{TP + TN}{All} = \frac{5 + 2}{10} = 70\%$$

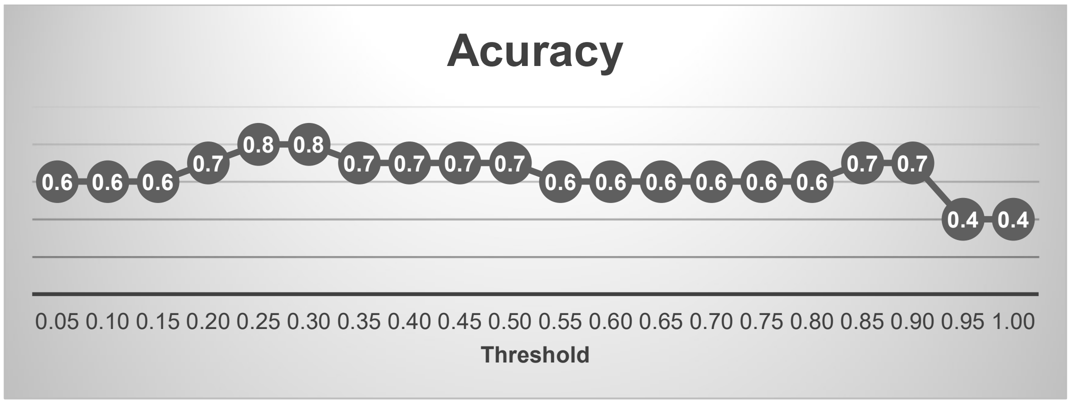
$specifity = \frac{TN}{N} = \frac{2}{4} = 50\%$ $\quad recall = \frac{TP}{P} = \frac{5}{6} = 83\%$ $\quad precision = \frac{TP}{TP+FP} = \frac{5}{6} = 83\%$

The trained classifier presents an 70% accuracy, albeit only 50% of the true negatives are perceived as so. Otherwise, the model predicts the true positives accurately, recall of 83%, and also does present a high percentage of false positives, precision of 83%.

However, considering the extremely reduced data set and that the training and test sets are the same, plus the assumptions that the numeric sets are normally distributed, the trained classifier is unlikely to perform as well on a larger sample of data.

4)

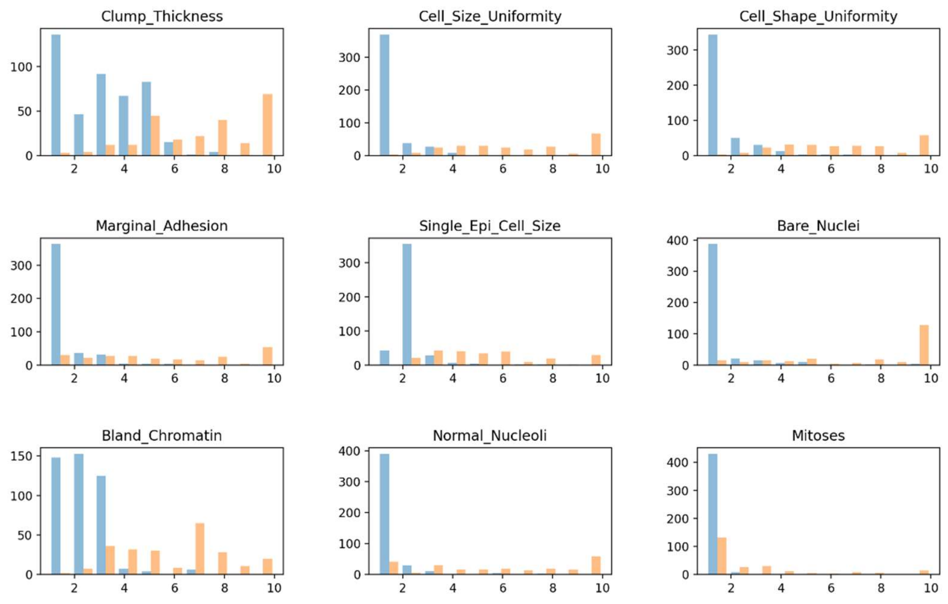
Calculating the accuracy again with increasing thresholds you obtain the following plot:



Analyzing the obtain data it's clear that the highest achievable accuracy is 80% and that it corresponds to a threshold between 0.25 and 0.30. Given the default threshold of 0.5 and that the optimal threshold for our classifier is around 0.3 our model is imbalanced.

II. Programming and critical analysis

5)



6)

Given the met conditions, using the appended code, the following results were obtained

Neighbors (k)	3	5	7
Mean Square error	0.0005140048829369539	0.0005295005027884739	0.0005891242124216451

Looking at the data, it becomes clear that the best number of neighbors to consider is 5, meaning k=5 is the least susceptible to the overfitting risk.

7)

Using a T-test related to evaluate the hypothesis "kNN is statistically superior to Naïve Bayes (multinomial assumption)" it was obtained that the p-value is 6.49695125239446e-05 which very close to 0 meaning that the tested hypothesis should be accepted. In other words, the results indicate that kNN is statistically superior to Naïve Bayes.

8)

Given the collected data, two reasons that could explain the difference in performance between kNN and Naïve Bayes are:

1. The breast.w.arff dataset has a considerable number of variables, that is known to hurt the performance of the Naïve Bayes, emphasizing the better performance of the kNN;
2. Furthermore, the variables not being independent will also affect classification, in the sense that it might lead to the zero-probability problem, meaning that that the conditional probability of a given variable might be zero, which will lead to an invalid prediction.

The presented reasons are confirmed by the results of the previous exercises, where the underperformance can be observed.

III. APPENDIX

```python
from scipy.io import arff
from scipy.stats import ttest_rel
import matplotlib.pyplot as plt
import numpy as np; import pandas as pd
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier

class Feature():
    def __init__(self, name, dataFrame):
        self.name, self.data = name, dataFrame[name]
    def traceConditional(self, Cond):
        # Split data conditionally
        class_0 = [x for i, x in enumerate(self.data) if Cond[i] == b'benign']
        class_1 = [x for i, x in enumerate(self.data) if Cond[i] != b'benign']
        # Build Histogram
        plt.hist([class_0, class_1], alpha=0.5, label="malignant", log=False)
        # plt.legend(loc='upper right')
        plt.title(self.name)

#KNN SEED
GROUPN = 39
# Extract Data
data = pd.DataFrame( arff.loadarff( "breast.w.arff" )[0] )

def quest5():
    # Create class index array
    arrayClass = data[data.columns[-1]]
    arrayFeatures = list()
    for feat in data.columns[0:-1]: arrayFeatures.append(Feature(feat, data))
    for i, feature in enumerate(arrayFeatures):
        plt.subplot(3, 3, i+1)
        feature.traceConditional(arrayClass)
    plt.tight_layout()
    plt.show()

X = data.drop(columns=data.columns[-1]).to_numpy().astype(int)
Y = data[data.columns[-1]].replace(b'benign', 0).replace(b'malignant', 1)

knn = StratifiedKFold(n_splits=10, random_state=GROUPN, shuffle=True)

def quest6():

    for k in [3, 5, 7]:
        knn_model = KNeighborsClassifier(n_neighbors=k)
        train_errors, test_errors = list(), list()

        for train_index, test_index in knn.split(X, Y):
            X_train, X_test = X[train_index], X[test_index]
            Y_train, Y_test = Y[train_index], Y[test_index]

            knn_model.fit(X_train, Y_train)

            Y_train_predict = knn_model.predict(X_train)
            Y_test_predict = knn_model.predict(X_test)

            fold_train_error = mean_absolute_error(Y_train, Y_train_predict)
            fold_test_error = mean_absolute_error(Y_test, Y_test_predict)

            train_errors.append(fold_train_error)
            test_errors.append(fold_test_error)
        # Calculate mse for each k, lower value -> less risk of overfitting
        err = mean_squared_error(train_errors, test_errors)
        print(f"Mean square error: {err} (k = {k})")

def quest7():

    knn_model = KNeighborsClassifier(n_neighbors=3)

    nbayes_model = MultinomialNB()

    knnScore, nbayesScore = list(), list()

    for train_index, test_index in knn.split(X, Y):
        X_train, X_test = X[train_index], X[test_index]
        Y_train, Y_test = Y[train_index], Y[test_index]

        knn_model.fit(X_train, Y_train); nbayes_model.fit(X_train, Y_train)
        knnScore.append(knn_model.score(X_test, Y_test))

        nbayesScore.append(nbayes_model.score(X_test, Y_test))

    stat, p_value = ttest_rel(knnScore, nbayesScore, alternative='greater'); print(f"p Value: {p_value}")

quest5(); quest6(); quest7()
```

END