**I. Pen-and-paper**

### Priors:

### Y1:

### Y2:

### Y3 & Y4:



Appling the trained model to the data set

|  |  |  |
| --- | --- | --- |
|  | P(Class = 0) | P(Class = 1) |
| x1 | 0.84 | 0.16 |
| x2 | 0.20 | 0.80 |
| x3 | 0.76 | 0.24 |
| x4 | 0.46 | 0.54 |
| x5 | 0.46 | 0.54 |
| x6 | 0.07 | 0.93 |
| x7 | 0.06 | 0.94 |
| x8 | 0.47 | 0.53 |
| x9 | 0.70 | 0.30 |
| x10 | 0.09 | 0.91 |

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

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.

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**

1. Graphical user interface, application

   Description automatically generated

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.

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.

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**

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**