**I. Pen-and-paper**

Prior : *Class 0*:  *Class 1*:

Y1 distribution:

* *Class 0*:
* *Class 1*: 0.2881

Y2 probability mass function:

* *Class 0*:
* *Class 1*:

Y3 and Y4 distribution:

* *Class 0*:
* *Class 1*:

Assuming naïve Bayes : , where the likelihood of each conditional variable to the class is given by the distributions calculated in question **1)**

Normalization:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x1 | | x2 | | x3 | | x4 | | x5 | |
| Class | c = 0 | c = 1 | c = 0 | c = 1 | c = 0 | c = 1 | c = 0 | c = 1 | c = 0 | c = 1 |
| P(class = c) | 0.4 | 0.6 | 0.4 | 0.6 | 0.4 | 0.6 | 0.4 | 0.6 | 0.4 | 0.6 |
| P(y1=Y1|class=c) | 0.569 | 0.224 | 1.374 | 1.364 | 1.639 | 1.209 | 1.374 | 1.364 | 1.639 | 0.950 |
| P(y2=Y2|class=c) | 0.5 | 0.1(6) | 0.25 | 0.(3) | 0.5 | 0.1(6) | 0.25 | 0.5 | 0.25 | 0.(3) |
| P(y3=Y3, y4=Y4 | class = c) | 1.207 | 1.210 | 0.460 | 0.955 | 0.707 | 0.610 | 0.512 | 0.203 | 1.174 | 1.206 |
| P(x | class = c) | 0.343 | 0.045 | 0.158 | 0.434 | 0.579 | 0.123 | 0.176 | 0.138 | 0.481 | 0.382 |
| P(class = c | x) | 1.373 | 0.271 | 0.633 | 2.605 | 2.317 | 0.738 | 0.704 | 0.829 | 1.925 | 2.293 |
| Normalization | 0.835 | 0.165 | 0.195 | 0.805 | 0.758 | 0.242 | 0.459 | 0.541 | 0.456 | 0.544 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x6 | | x7 | | x8 | | x9 | | x10 | |
| Class | c = 0 | c = 1 | c = 0 | c = 1 | c = 0 | c = 1 | c = 0 | c = 1 | c = 0 | c = 1 |
| P(class = c) | 0.4 | 0.6 | 0.4 | 0.6 | 0.4 | 0.6 | 0.4 | 0.6 | 0.4 | 0.6 |
| P(y1=Y1|class=c) | 0.569 | 1.209 | 0.116 | 0.662 | 1.639 | 1.209 | 1.374 | 0.662 | 0.280 | 0.950 |
| P(y2=Y2|class=c) | 0.25 | 0.5 | 0.25 | 0.5 | 0.25 | 0.(3) | 0.5 | 0.1(6) | 0.25 | 0.5 |
| P(y3=Y3, y4=Y4 | class = c) | 0.334 | 0.672 | 0.707 | 0.610 | 1.085 | 0.838 | 0.217 | 0.387 | 1.080 | 1.123 |
| P(x | class = c) | 0.047 | 0.406 | 0.021 | 0.202 | 0.445 | 0.338 | 0.149 | 0.043 | 0.076 | 0.534 |
| P(class = c | x) | 0.190 | 2.437 | 0.082 | 1.212 | 1.778 | 2.027 | 0.598 | 0.256 | 0.303 | 3.202 |
| Normalization | 0.072 | 0.928 | 0.063 | 0.937 | 0.467 | 0.533 | 0.700 | 0.300 | 0.087 | 0.913 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | Sum |
| 0 | 2 | 2 | 4 |
| 1 | 1 | 5 | 6 |
| Sum | 3 | 7 | 10 |

2. We used the posteriors as thresholds, because between them, the obtained results from the classifier are the same. So, we can conclude that the best threshold to use is where . Having a large threshold is not ideal, because the dataset is too small and the class results are not balanced (40/60), predicting more 1’s is going to lead to a better accuracy. So, the threshold that optimizes training accuracy will be approximately

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| True | P(class=0|x) | 0.06344 | 0.07224 | 0.08653 | 0.19538 | 0.45643 | 0.45923 | 0.46736 | 0.70014 | 0.75846 | 0.83522 |
| 0 | 0.835224838 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0.195377549 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0.758462001 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0.459234927 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0.456434993 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0.072240527 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0.063438589 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0.467357063 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0.700142029 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0.086525992 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| accuracy | | 0.5 | 0.6 | 0.7 | 0.6 | 0.7 | 0.6 | 0.7 | 0.8 | 0.7 | 0.6 |

**II. Programming and critical analysis**

Chart

Description automatically generated5)

1. Using 10-fold cross validation we separated the given dataset into 10 subsets that are used to train (9 subsets) and test (1 subset) and repeat this for all combinations of train and test data. The test phase consists in predicting the result and then compare it to the real one, calculating the accuracy of it. The average accuracy for 𝑘 ∈ {3,5,7} is respectively {0.970801364, 0.975191816, 0.972271952}

From the obtained data, we can conclude that 𝑘 = 5 is less susceptible to the overfitting risk, as it is the value that register the higher value of accuracy, but theoretically having a bigger k reduces the spatial dependency leading to a lower overfitting risk.

1. Applying the T-Student formula for the hypothesis H1: “𝑘NN is statistically superior to Naïve Bayes", and H0 being the null hypothesis, we’ll get the following values:

p-value = 0.0001196872339771996 statistic=5.864575247914318

In conclusion, since the p-value is lower than 0.05, it indicates strong evidence against the null hypothesis, validating the statement of H1.

1. Differences in performance between 𝑘NN and Naïve Bayes can be caused by:

* Naïve Bayes makes the assumption that all variables are independent, and there is no guarantee that this is the case in this context.
* kNN is a simple algorithm when compared to Naïve Bayes, but it is quite accurate when handling datasets of small samples, even more than Naïve Bayes, that requires a larger, more complex set of data to be more accurate.

**III. APPENDIX**

5)

dataset = pandas.DataFrame(arff.loadarff(<path>)[0]) ; lines = len(dataset["Bare\_Nuclei"])

# Loop through all variables

for variable in variables:

benign = []; malignant = []

for i in range(lines):

if (dataset["Class"][i] == b'malignant'):

malignant.append(dataset[variable][i])

elif (dataset["Class"][i] == b'benign'):

benign.append(dataset[variable][i])

labels = ["benign", "malignant"]

plt.hist([benign, malignant] , bins=bins, density=True, alpha=0.5, align="left")

plt.savefig(<path>)

plt.clf()

6)

inputs = dataset.drop(columns=["Class"]).values; outputs = dataset["Class"].values

k\_fold = KFold(n\_splits=10, random\_state=13, shuffle=True);accuracyKnn = {"3": [], "5": [], "7": []}

for numNeighbors in range(3, 8, 2):

for train, test in k\_fold.split(dataset):

knnClassifier = KNeighborsClassifier(n\_neighbors=numNeighbors, weights="uniform")

knnClassifier = knnClassifier.fit(inputs[train], outputs[train])

accuracyKnn[str(numNeighbors)].append(knnClassifier.score(inputs[test], outputs[test])

for numNeighbors in accuracyKnn:

averageAccuracy = sum(accuracyKnn [numNeighbors]) / len(accuracyKnn[numNeighbors])

print(f"average accuracy k={numNeighbors} => {averageAccuracy}")

7)

accuracyNB = []

for train, test in k\_fold.split(dataset):

multinomialNBClassifier = MultinomialNB()

multinomialNBClassifier = multinomialNBClassifier.fit(inputs[train], outputs[train])

accuracyNB.append(multinomialNBClassifier.score(inputs[test], outputs[test])

pValue = ttest\_rel(accuracykNN3, accuracyNB, alternative="greater").pvalue