Aprendizagem 2022

Homework II – Group 019

Diogo Gaspar 99207, Rafael Oliveira 99311

Part I: Pen and paper

1. **Compute the recall of a distance-weighted kNN with $k = 5$ and distance $d(x_1, x_2) = \text{Hamming}(x_1, x_2) + \frac{1}{2}$ using leave-one-out evaluation schema (i.e., when classifying one observation, use all remaining ones).**

   For starters, it's worth noting that, in this context, the **Hamming distance** between two observations x_1 and x_2 is defined as the number of attributes that differ between them.

   Knowing this, we can now create an 8×8 matrix (as can be seen below), where each entry represents the Hamming distance ($+\frac{1}{2}$) between two observations. This matrix is symmetric, of course. Each column i, here, will have $8 - 1 = 7$ associated entries, each representing the distance d between the observation x_i and the remaining 7 observations: we will, then, pick the $k = 5$ nearest neighbors according to said distance, classifying x_i in a **distance-weighted** manner.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
x_1	×	5/2	3/2	1/2	3/2	3/2	3/2	5/2
x_2	5/2	×	3/2	5/2	3/2	3/2	3/2	1/2
x_3	3/2	3/2	×	3/2	5/2	5/2	1/2	3/2
x_4	1/2	5/2	3/2	×	3/2	3/2	3/2	5/2
x_5	3/2	3/2	5/2	3/2	×	1/2	5/2	3/2
x_6	3/2	3/2	5/2	3/2	1/2	×	5/2	3/2
x_7	3/2	3/2	1/2	3/2	5/2	5/2	×	3/2
x_8	5/2	1/2	3/2	5/2	3/2	3/2	3/2	×

Table 1: Distance d between observations - in teal, a given observation's k nearest neighbors

For each observation x_i, the k nearest neighbors are, therefore, the ones represented in teal in the table above. Instead of predicting a class given the neighbors' mode, we'll want to choose it in a distance-weighted manner here: this means that, for each observation x_i, we'll want to compute the **weighted majority vote** of its k nearest neighbors, where the weight of each neighbor is given by:

$$w_{i,j} = \frac{1}{d(x_i, x_j)}$$

where x_j is one of the k nearest neighbors of x_i. Considering the data gathered up until now, we'll have the following conclusions regarding the model's classification for each instance:

	Weighted distance to N	Weighted distance to P	Predicted Class
x_1	$3 * 1/(3/2) = 2$	$1/(3/2) + 1/(1/2) = 8/3$	P
x_2	$3 * 1/(3/2) + 1/(1/2) = 4$	$1/(3/2) = 2/3$	N
x_3	$1/(3/2) + 1/(1/2) = 8/3$	$3 * 1/(3/2) = 2$	N
x_4	$3 * 1/(3/2) = 2$	$1/(3/2) + 1/(1/2) = 8/3$	P
x_5	$1/(3/2) + 1/(1/2) = 8/3$	$3 * 1/(3/2) = 2$	N
x_6	$1/(3/2) + 1/(1/2) = 8/3$	$3 * 1/(3/2) = 2$	N
x_7	$2 * 1/(3/2) = 1/3$	$3 * 1/(3/2) + 1/(1/2) = 4$	P
x_8	$3 * 1/(3/2) = 2$	$1/(3/2) + 1/(1/2) = 8/3$	P

Table 2: Distance weighting for each observation

We'll have, given the data gathered above, the following confusion matrix:

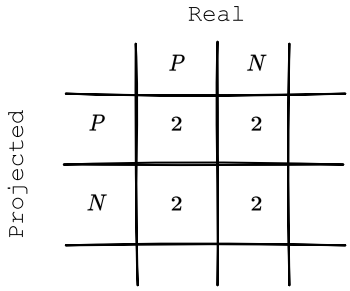

Figure 1: Confusion Matrix

Moreover, the **recall** of a classifier is defined as the ratio between the number of true positives and the number of true positives plus the number of false negatives that the classifier makes. Looking at the confusion matrix above, we can assert that the associated recall will, therefore, be:

$$\frac{TP}{TP + FN} = \frac{2}{2+2} = \frac{2}{4} = 0.5$$

2. **Considering the nine training observations, learn a Bayesian classifier assuming: i) y_1 and y_2 are dependent, ii) $\{y_1, y_2\}$ and $\{y_3\}$ variable sets are independent and equally important, and iii) y_3 is normally distributed. Show all parameters.**

   Considering both variable sets, $\{y_1, y_2\}$ and $\{y_3\}$, to be independent and equally important, it'll make sense to train a Naive Bayes classifier here, such that (and utilizing Bayes' theorem):

$$P(C =_P^N |y_1, y_2, y_3) = \frac{P(y_1, y_2, y_3|C =_P^N)P(C =_P^N)}{P(y_1, y_2, y_3)}$$

   More so, since $\{y_1, y_2\}$ and $\{y_3\}$ are independent (and y_1 and y_2 are dependent), we can rewrite the above as:

$$P(C =_P^N |y_1, y_2, y_3) = \frac{P(y_1, y_2|C =_P^N)P(y_3|C =_P^N)P(C =_P^N)}{P(y_1, y_2)P(y_3)}$$

   A Bayesian classifier's goal here will be to maximize the numerator of the above equation, since we'll want to find the "maximum posterior hypothesis" (i.e $P(C =_P^N)$, here), and such a maximization will be independent of $P(y_1, y_2, y_3)$. As such, we know that we'll be looking for the following:

$$\text{argmax}_{c\in\{N,P\}} P(y_1, y_2|C = c)P(y_3|C = c)P(C = c)$$

   For starters, we can note that, from the given training set:

$$P(C = N) = \frac{4}{9}, \quad P(C = P) = \frac{5}{9}$$

   We also know that y_3 is normally distributed, meaning we'll have:

$$P(y_3|C =_P^N) \sim \mathcal{N}(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

   By also looking at the given training set (which includes the new, ninth sample), we'll be able to extrapolate the following probabilities:

$C = N : P(C = N) = \frac{4}{9}$

- $P(y_1 = A, y_2 = 0 | C = N) = 0$
- $P(y_1 = A, y_2 = 1 | C = N) = \frac{1}{4}$
- $P(y_1 = B, y_2 = 0 | C = N) = \frac{2}{4} = \frac{1}{2}$
- $P(y_1 = B, y_2 = 1 | C = N) = \frac{1}{4}$

Regarding y_3 and N labeled observations, we'll have the following parameters:

$$\mu = \frac{1 + 0.9 + 1.2 + 0.8}{4} = 0.975$$

$$\sigma^2 = \frac{1}{4 - 1} \sum_{i=1}^{4} (y_{3,i} - \mu)^2 = 0.029$$

Therefore:

$$P(y_3 | C = N) \sim \mathcal{N}(x | 0.975, 0.029)$$

$C = P : P(C = P) = \frac{5}{9}$

- $P(y_1 = A, y_2 = 0 | C = P) = \frac{2}{5}$
- $P(y_1 = A, y_2 = 1 | C = P) = \frac{1}{5}$
- $P(y_1 = B, y_2 = 0 | C = P) = \frac{1}{5}$
- $P(y_1 = B, y_2 = 1 | C = P) = \frac{1}{5}$

Regarding y_3 and P labeled observations, we'll have the following parameters:

$$\mu = \frac{1.2 + 0.8 + 0.5 + 0.9 + 0.8}{5} = 0.84$$

$$\sigma^2 = \frac{1}{5 - 1} \sum_{i=1}^{5} (y_{3,i} - \mu)^2 = 0.063$$

Therefore:

$$P(y_3 | C = P) \sim \mathcal{N}(x | 0.84, 0.063)$$

The model is now ready to be used to classify new observations.

3. **Under a MAP assumption, compute $P(Positive|x)$ of each testing observation.**

   Considering Bayes' theorem, we know $P(Positive|x)$ can also be written as follows:

$$P(Positive|x) = \frac{P(x|Positive)P(Positive)}{P(x)}$$

   Since x has two independent variable sets, $\{y_1, y_2\}$ and $\{y_3\}$, we'll be able to write the above expression as follows (note that x's y_1 value will be written as y_1, and so on, for simplicity's sake):

$$P(Positive|x) = \frac{P(y_1, y_2|Positive)P(y_3|Positive)P(Positive)}{P(y_1, y_2)P(y_3)}$$

   Let's consider the three samples given in the question's statement:

$$x'_1 = \begin{pmatrix} A \\ 1 \\ 0.8 \end{pmatrix}, \quad x'_2 = \begin{pmatrix} B \\ 1 \\ 1 \end{pmatrix}, \quad x'_3 = \begin{pmatrix} B \\ 0 \\ 0.9 \end{pmatrix}$$

   Considering the training observations, we can gather that there are:

x'_1	x'_2	x'_3
Among all (five) positive training samples, one with features $y_1 = A, y_2 = 1$, two with feature $y_3 = 0.8$. Among all (nine) training samples, two with features $y_1 = A, y_2 = 1$, three with feature $y_3 = 0.8$.	Among all (five) positive training samples, one with features $y_1 = B, y_2 = 1$, zero with feature $y_3 = 1$. Among all (nine) training samples, two with features $y_1 = B, y_2 = 1$, one with feature $y_3 = 1$.	Among all (five) positive training samples, one with features $y_1 = B, y_2 = 0$, one with feature $y_3 = 0.9$. Among all (nine) training samples, three with features $y_1 = B, y_2 = 0$, two with feature $y_3 = 0.9$.

   Note, of course, that the probabilities regarding y_1 and y_2 features may be calculated in a discrete manner, directly utilizing the values gathered above. The same can't be said for y_3 with it being normally distributed: we'll need to calculate the parameters for it (note that Positive parameters were computed in the previous question), and only then will we be able to calculate the needed probabilities regarding these samples' y_3 values:

$$\mu_{y_3,Positive} = 0.84, \quad \mu_{y_3} = \frac{(1.2 + 0.8 + \cdots + 0.8)}{9} = 0.9 \tag{1}$$

$$\sigma^2_{y_3,Positive} = 0.063, \quad \sigma^2_{y_3} = \frac{1}{9-1}\sum_{i=1}^{9}(y_{3,i} - \mu)^2 = 0.0475 \tag{2}$$

With y_3 being normally distributed, we can calculate both $P(y_3|Positive)$ and $P(y_3)$ using the following formulas:

$$P(y_3|Positive) = \frac{1}{\sqrt{2\pi\sigma^2_{y_3,Positive}}} \exp\left(-\frac{(y_3 - \mu_{y_3,Positive})^2}{2\sigma^2_{y_3,Positive}}\right)$$

$$P(y_3) = \frac{1}{\sqrt{2\pi\sigma^2_{y_3}}} \exp\left(-\frac{(y_3 - \mu_{y_3})^2}{2\sigma^2_{y_3}}\right)$$

x'_1	x'_2	x'_3
$P(y_3 = 0.8\|P) \approx 1.569,$ $P(y_3 = 0.8) \approx 1.6476$	$P(y_3 = 1\|P) = 1.2972,$ $P(y_3 = 1) = 1.6476$	$P(y_3 = 0.9\|P) = 1.5447,$ $P(y_3 = 0.9) = 1.8305$

We can, therefore, assert that:

$$P(Positive|x'_1) = \frac{1/5 \times 1.569 \times 5/9}{2/9 \times 1.6476} = 0.4763$$

$$P(Positive|x'_2) = \frac{1/5 \times 1.2972 \times 5/9}{2/9 \times 1.6476} = 0.3937$$

$$P(Positive|x'_3) = \frac{1/5 \times 1.5447 \times 5/9}{3/9 \times 1.8305} = 0.2813$$

4. **Given a binary class variable, the default decision threshold of $\theta = 0.5$,**

$$f(x|\theta) = \begin{cases} Positive & \textbf{if } P(positive|x) > \theta \\ Negative & otherwise \end{cases}$$

can be adjusted. Which decision threshold – 0.3, 0.5 or 0.7 – optimizes testing accuracy?

As given by the question's statement, we know the actual class values of the testing observations. Moreover, we've calculated $P(Positive|x)$ for each of them in the previous question's answer, so we can easily calculate the accuracy of the classifier for each of the three decision thresholds (considering only these three testing samples):

	Class	$P(Positive\|x)$	$\theta = 0.3$	$\theta = 0.5$	$\theta = 0.7$
x'_1	*Positive*	0.4763	*Positive*	*Negative*	*Negative*
x'_2	*Positive*	0.3937	*Positive*	*Negative*	*Negative*
x'_3	*Negative*	0.2813	*Negative*	*Negative*	*Negative*

Table 3: $f(x|\theta)$ for varying thresholds, given three testing samples

Here, the classifier's accuracy, considering the table above, amounts to:

$$\begin{aligned} \text{Accuracy}(\theta = 0.3) &= 1 \\ \text{Accuracy}(\theta = 0.5) &= 1/3 \\ \text{Accuracy}(\theta = 0.7) &= 1/3 \end{aligned}$$

As can be seen above, a threshold of $\theta = 0.3$ yields a higher accuracy than both 0.5 and 0.7, hence it should be chosen in order to optimize testing accuracy.

Part II: Programming

The code utilized in order to answer this section's first two questions answers can be found in this report's appendix. There's both a "general" code section, which can be found initially, and two functions, `first` and `second`, utilized to answer the respective questions - the first to plot the confusion matrices, while the second to test the given hypothesis.

5. **Using `sklearn`, considering a 10-fold stratified cross validation (`random=0`), plot the cumulative testing confusion matrices of kNN (uniform weights, $k = 5$, Euclidean distance) and Naïve Bayes (Gaussian assumption). Use all remaining classifier parameters as default.**

   We ended up utilizing `seaborn`'s `heatmap` function to plot the required confusion matrices side-by-side.

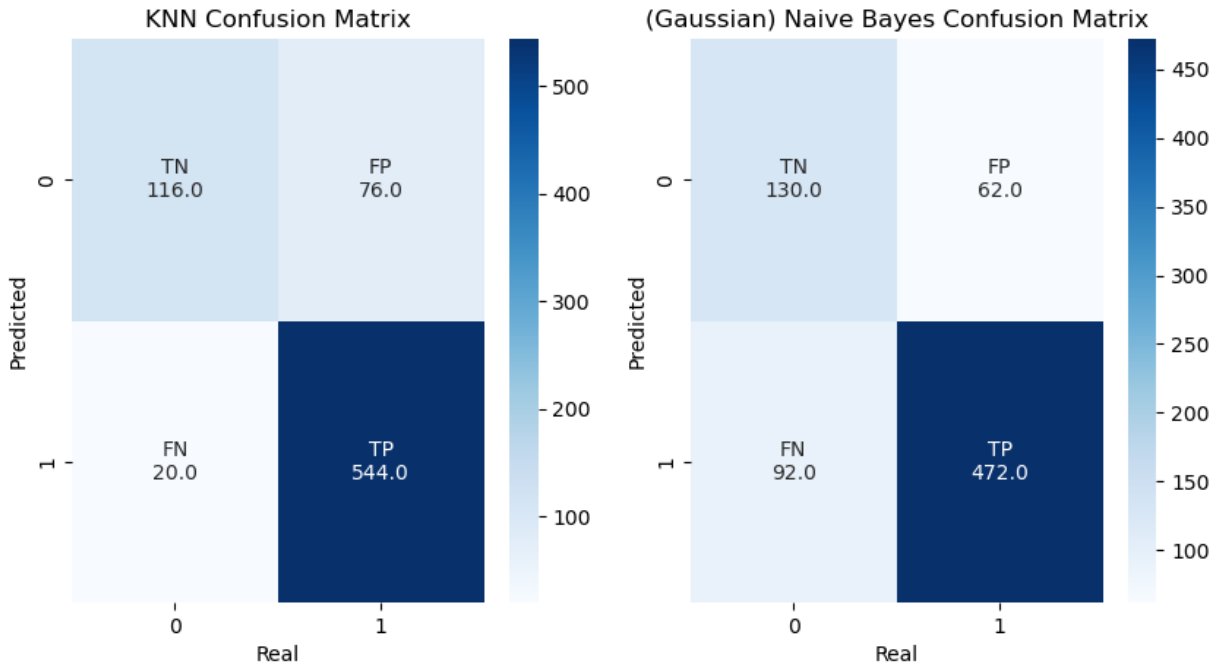

Figure 2: Cumulative testing confusion matrices of kNN and Naïve Bayes

As a relevant note, we opted to **normalize** our data, utilizing `sklearn`'s `StandardScaler`, in order to avoid overfitting the data (even more so while having relatively small training and testing sample subsets). As such, both this question's answer **and the following ones** will be based on normalized data.

6. **Using `scipy`, test the hypothesis "kNN is statistically superior to Naïve Bayes regarding accuracy", asserting whether it is true.**

   `scipy.stats` provides the `ttest_rel` function, which can be utilized to test a hypothesis given two related samples - here, both kNN's and Naïve Bayes' scores are inherently related, since we're working with the same dataset on both classifiers, hence we can use this function to test the hypothesis.

   We opted to utilize it with a `alternative = 'greater'` parameter, since we're interested in testing whether kNN's accuracy is statistically superior to Naïve Bayes' accuracy, and not the other way around - as such, a "right-tailed" test is more appropriate. The considered hypotheses are, therefore:

   - H_0: kNN's accuracy is statistically equal to Naïve Bayes'
   - H_1: kNN's accuracy is statistically superior to Naïve Bayes'

   As a side-note, we've considered, in absence of a given confidence level in the question's statement, a confidence level of $1 - \alpha = 0.95$. After performing the test, we obtained a *p-value* of ≈ 0.0013168 and a *t-statistic* of ≈ 4.1109, which leads us to assert that, given $\alpha = 0.05$, we should reject the null hypothesis and accept the alternative one - kNN's accuracy is statistically superior to Naïve Bayes' accuracy.

7. **Enumerate three possible reasons that could underlie the observed differences in predictive accuracy between kNN and Naïve Bayes.**

   Note: the homework's FAQ states that the answer could mention as low as two reasons, even though the original question's statement asks for three. We opted to answer the question as stated in the FAQ.

   In the previous question, we were able to assert that, regarding accuracy, kNN is indeed statistically superior to Naïve Bayes. There are a couple of reasons that could underlie this observation:

   - **Naïve Bayes' assumption of independence between features.**
     Naïve Bayes' assumption of independence between features is, in fact, a very strong assumption, which is not always true - more so in a relatively small dataset with a relatively large amount of features, such as the one we're working with. As such, Naïve Bayes' assumption of independence between features could be detrimental to its accuracy, since it could lead to a loss of information regarding the dataset's features.
   - **Unknown distribution of the dataset's features.**
     Naïve Bayes' assumption of Gaussian distribution of the dataset's features is, again, a very strong assumption, which is not always true - once again, even more prevalent in a relatively small dataset with a relatively large amount of features, such as this one. As such, by assuming a Gaussian distribution for **all** of the dataset's features, Naïve Bayes is very likely to underfit the dataset, since it's not taking into account the dataset's features' distributions, which could be very different from a Gaussian one.

Appendix

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import StratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from scipy.io.arff import loadarff
from scipy.stats import ttest_rel
from sklearn.preprocessing import StandardScaler

# Load data
data = loadarff('data/pd_speech.arff')
df = pd.DataFrame(data[0])
df['class'] = df['class'].str.decode('utf-8')

X = df.drop('class', axis=1)
y = df['class']

# Create 10-fold stratified cross validation
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=0)

# Create classifiers
knn = KNeighborsClassifier(n_neighbors=5, weights='uniform', metric='euclidean')
nb = GaussianNB()

knn_cm = np.zeros((2, 2))
nb_cm = np.zeros((2, 2))

knn_scores = np.array([])
nb_scores = np.array([])

for train_index, test_index in skf.split(X, y):
  X_train, X_test = X.iloc[train_index], X.iloc[test_index]
  y_train, y_test = y.iloc[train_index], y.iloc[test_index]

  # Normalize data
  scaler = StandardScaler().fit(X_train)
  X_train = scaler.transform(X_train)
  X_test = scaler.transform(X_test)

  knn.fit(X_train, y_train)
  nb.fit(X_train, y_train)

  knn_pred = knn.predict(X_test)
  nb_pred = nb.predict(X_test)

  # used in the first question
  knn_cm += confusion_matrix(y_test, knn_pred)
  nb_cm += confusion_matrix(y_test, nb_pred)

  # used in the second question
```

```
  knn_scores = np.append(knn_scores, accuracy_score(y_test, knn_pred))
  nb_scores = np.append(nb_scores, accuracy_score(y_test, nb_pred))

def first():
  """Used to answer the first question"""
  def create_plot(axis, labels, title, cm):
    plot = sns.heatmap(cm, annot=labels, cmap='Blues', ax=axis, fmt='s')
    plot.set_title(title)
    plot.set_ylabel('Predicted')
    plot.set_xlabel('Real')
    return plot

  def label_heatmap(cm):
    return np.asarray(
      [f"{name}\n{count}" for name, count in zip(GROUP_NAMES, cm.flatten())]
    ).reshape(2, 2)

  GROUP_NAMES =['TN', 'FP', 'FN', 'TP']
  KNN_LABELS = label_heatmap(knn_cm)
  NB_LABELS = label_heatmap(nb_cm)
  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))
  p1 = create_plot(ax1, KNN_LABELS, 'KNN Confusion Matrix', knn_cm)
  p2 = create_plot(ax2, NB_LABELS, '(Gaussian) Naive Bayes Confusion Matrix', nb_cm
   )
  plt.show()
  # save the plot to a file
  fig.savefig('assets/hw2-2.1.png')

def second():
  """Used to answer the second question"""
  return ttest_rel(knn_scores, nb_scores, alternative='greater')

# First programming question
first()

# Second programming question
results = second()
print("KNN better than NB?\np-value = {}\nt-statistic = {}".format(results.pvalue,
    results.statistic))
```