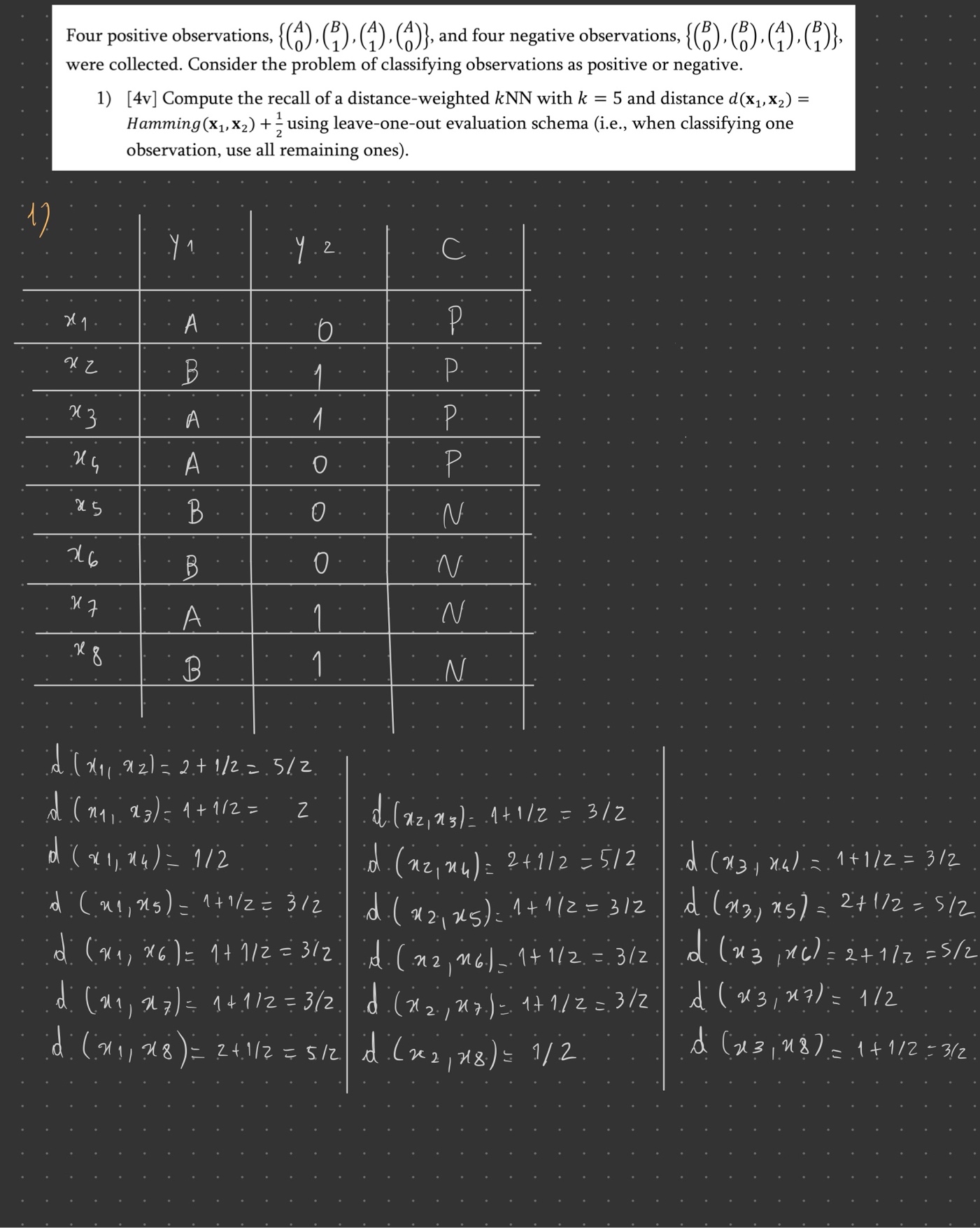
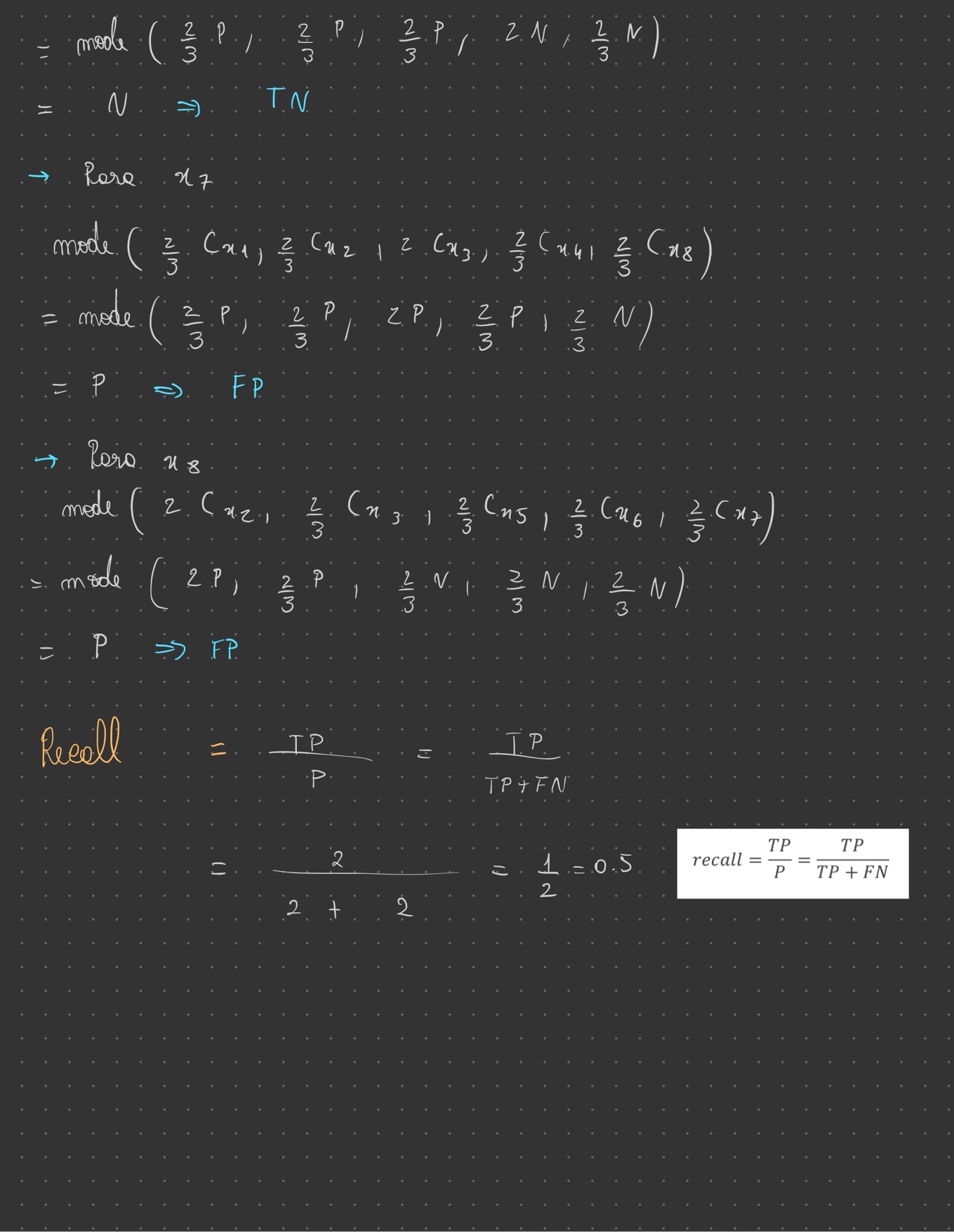
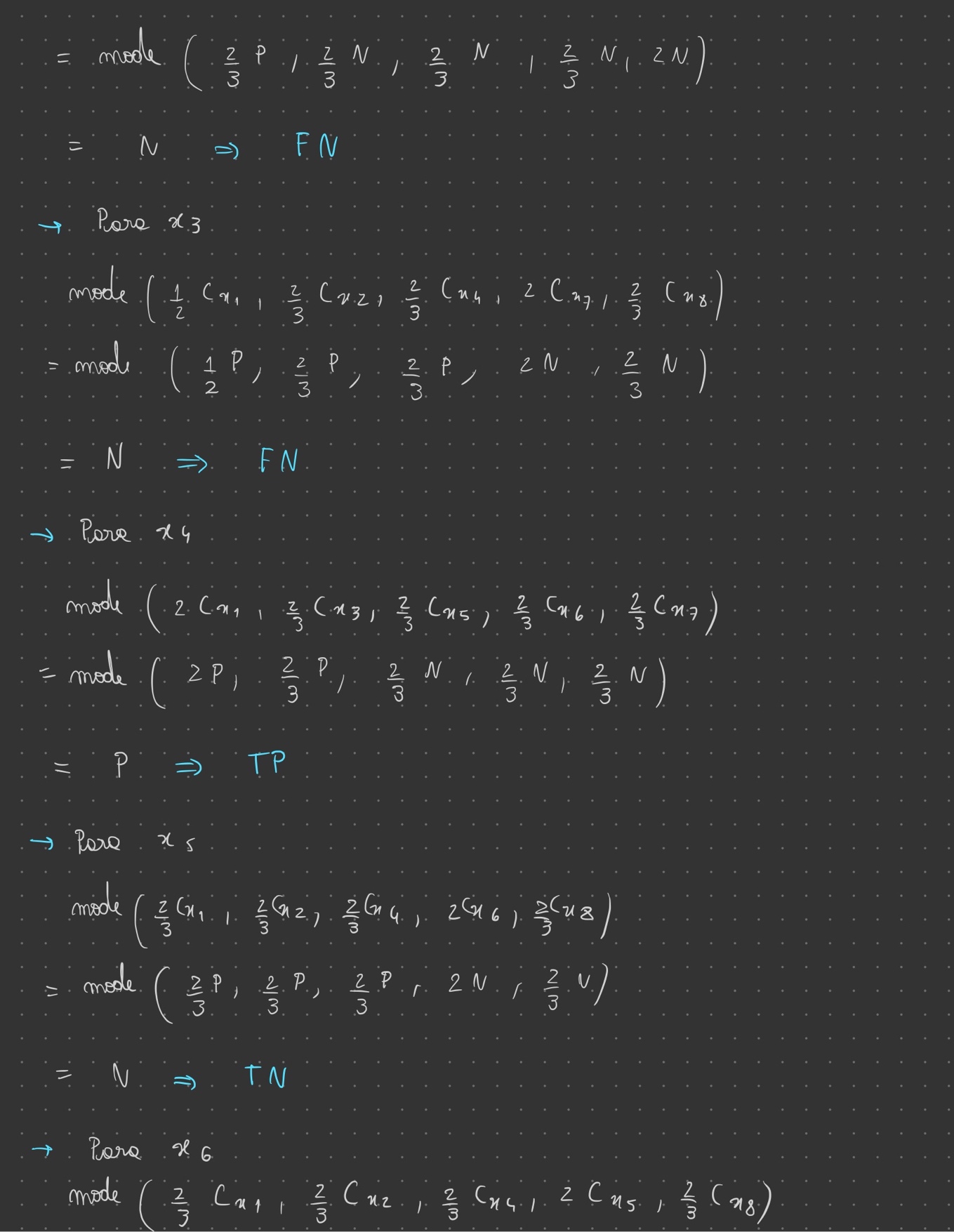
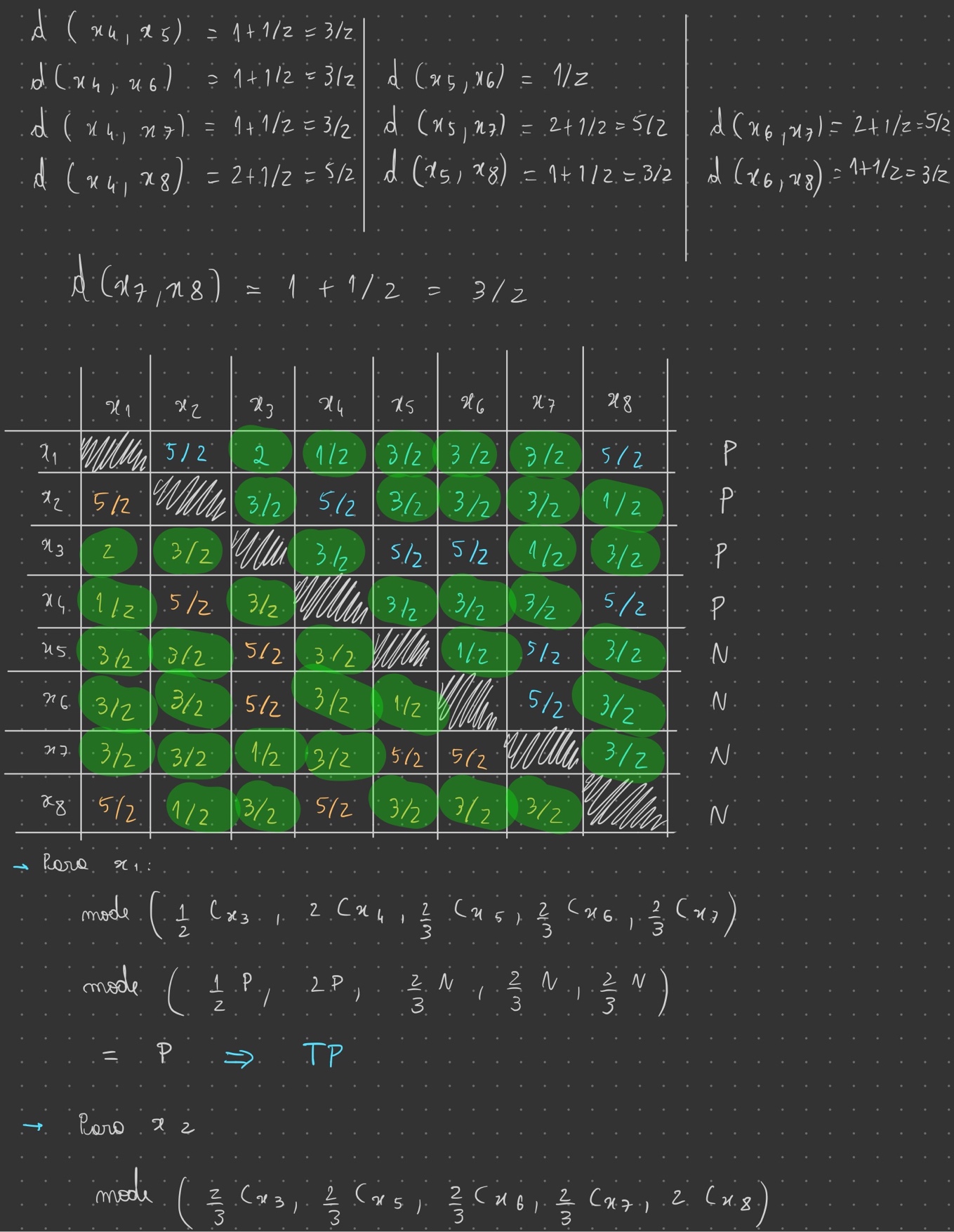
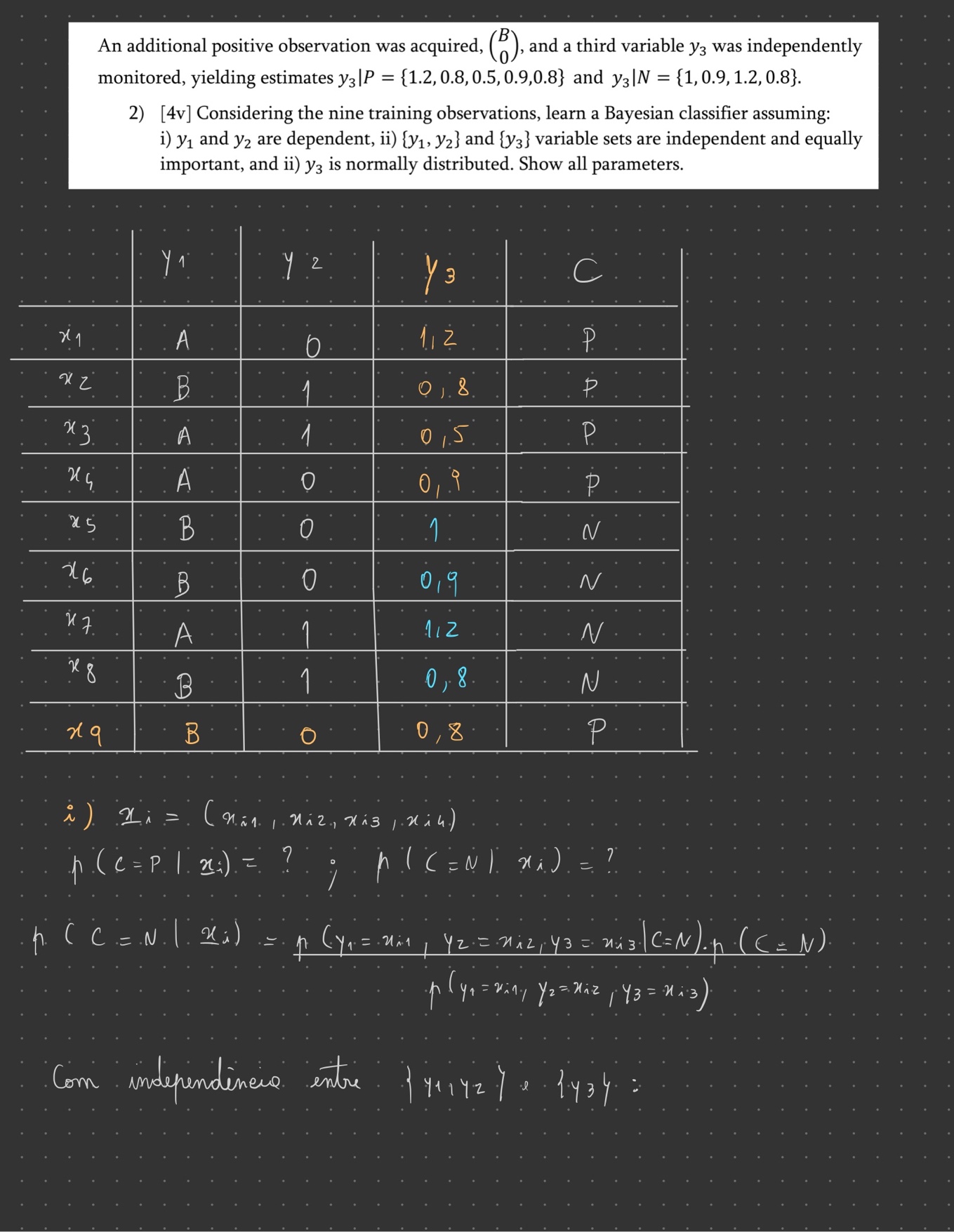
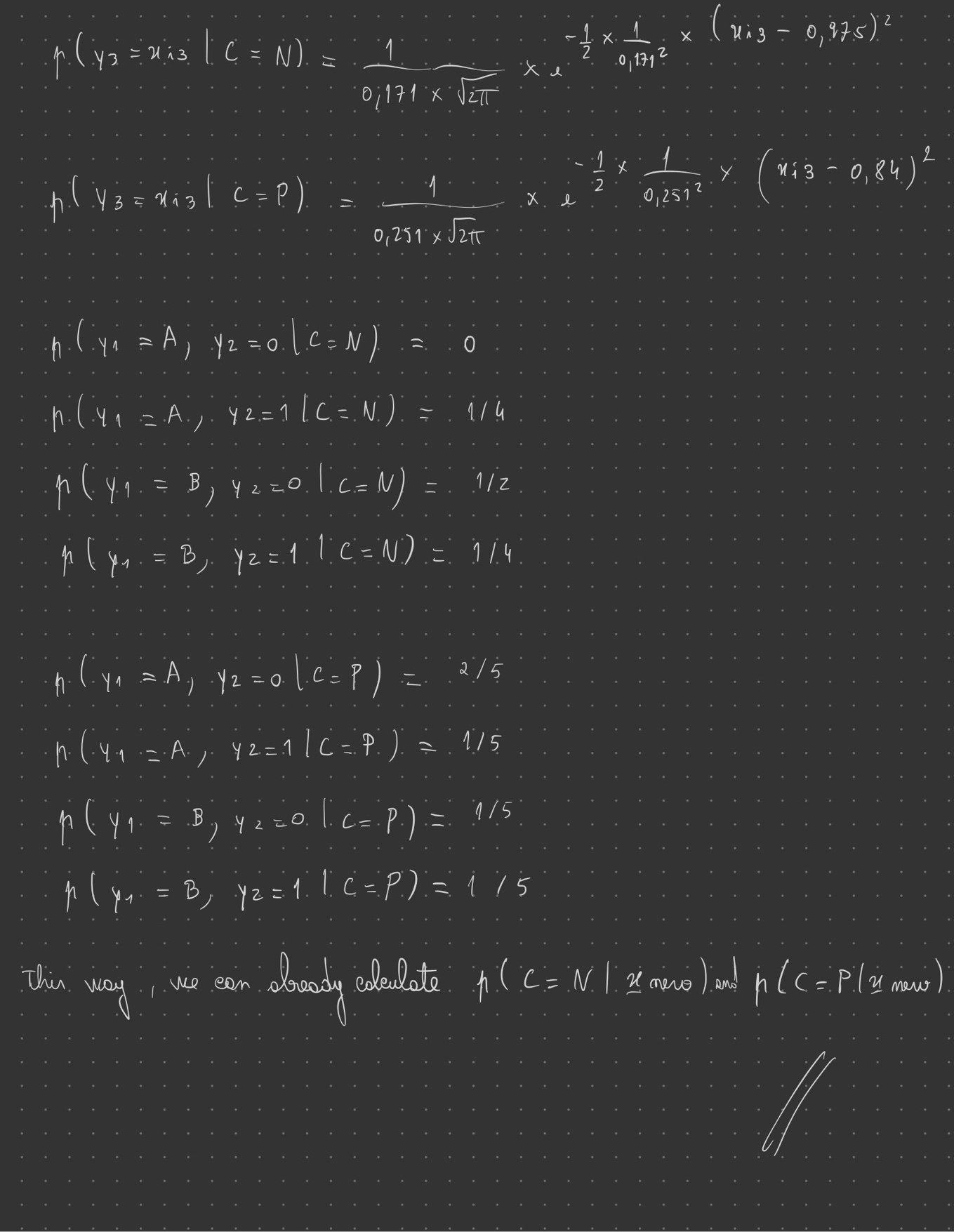
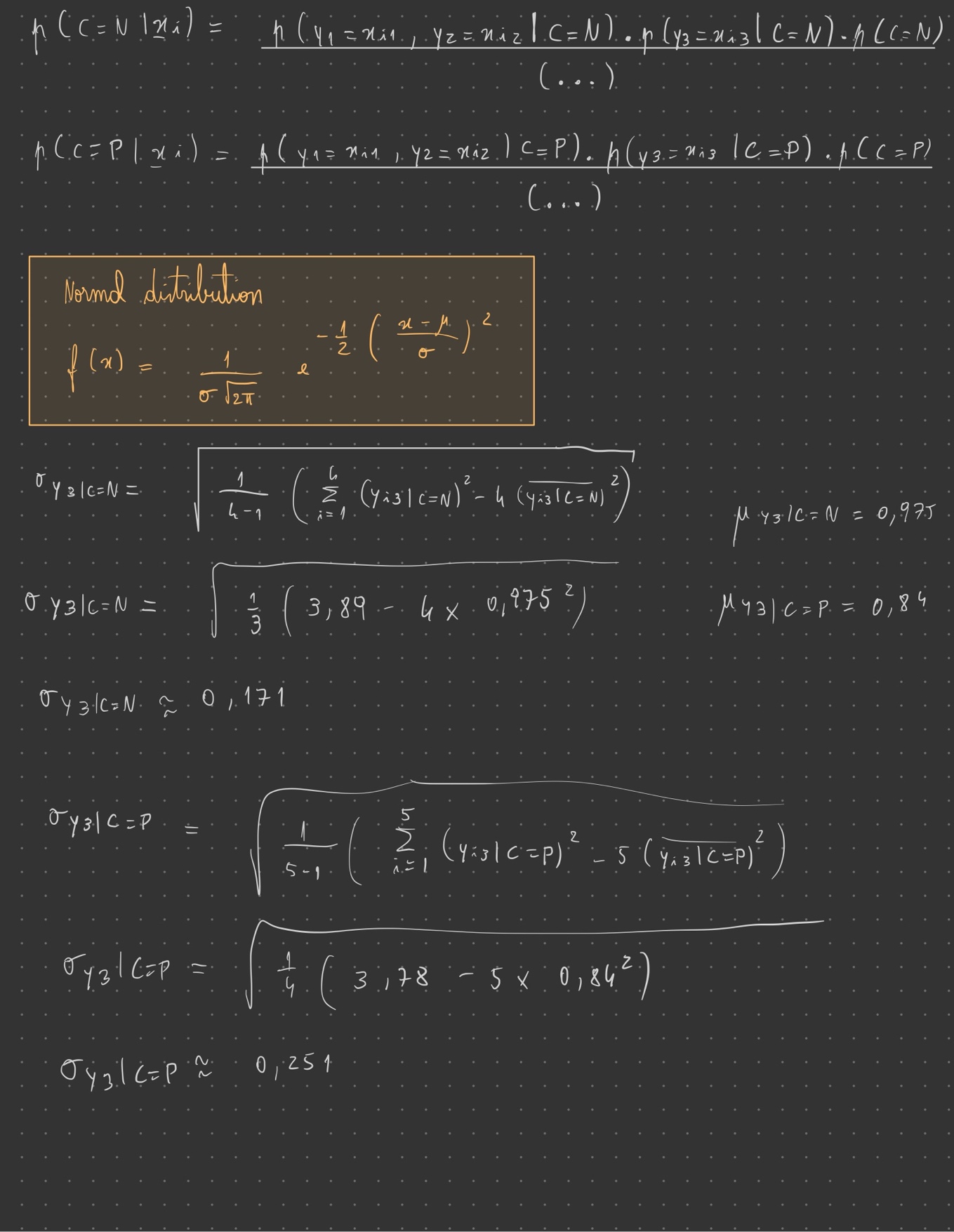
1. **Pen-and-paper**

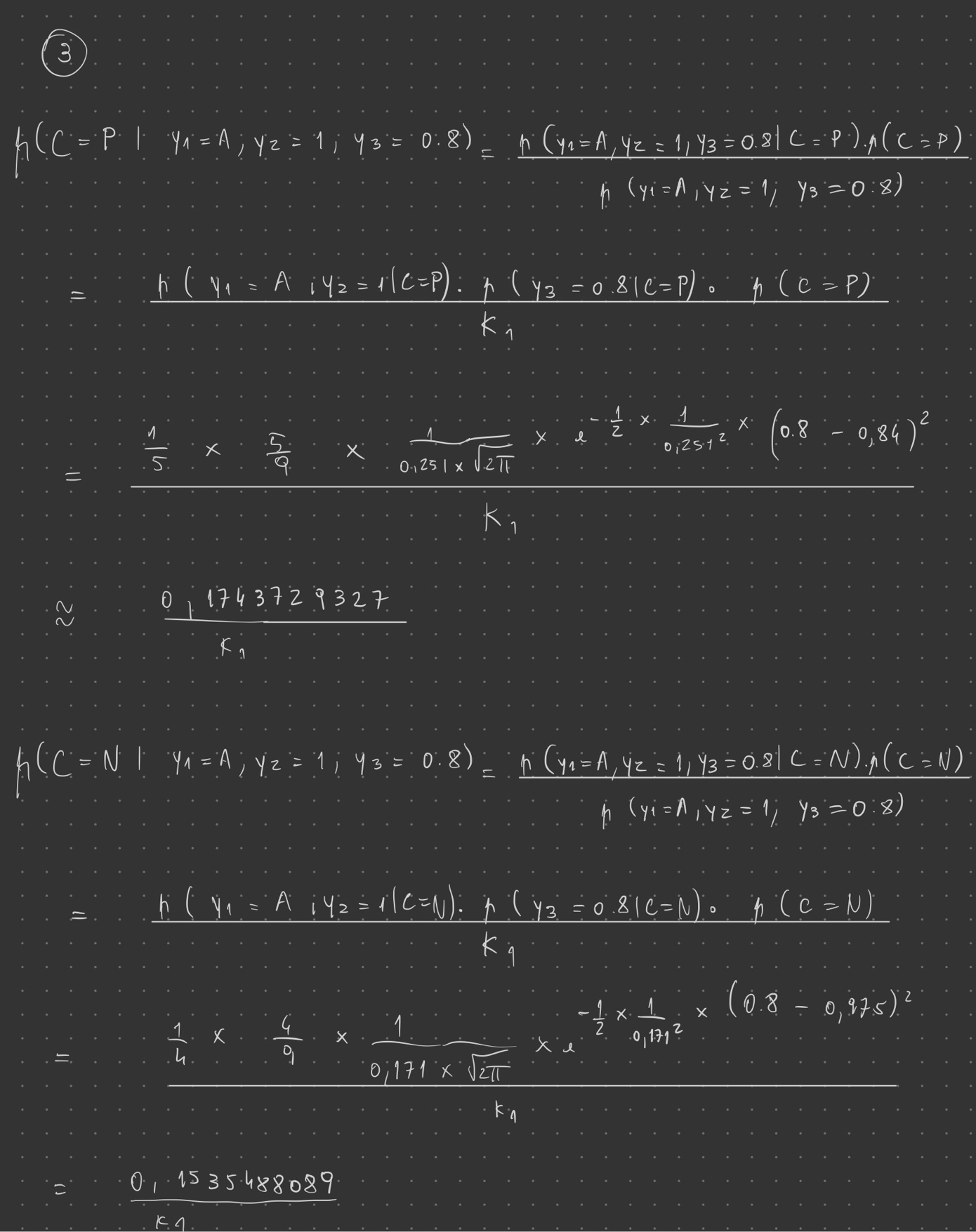
**1) Answer 1**

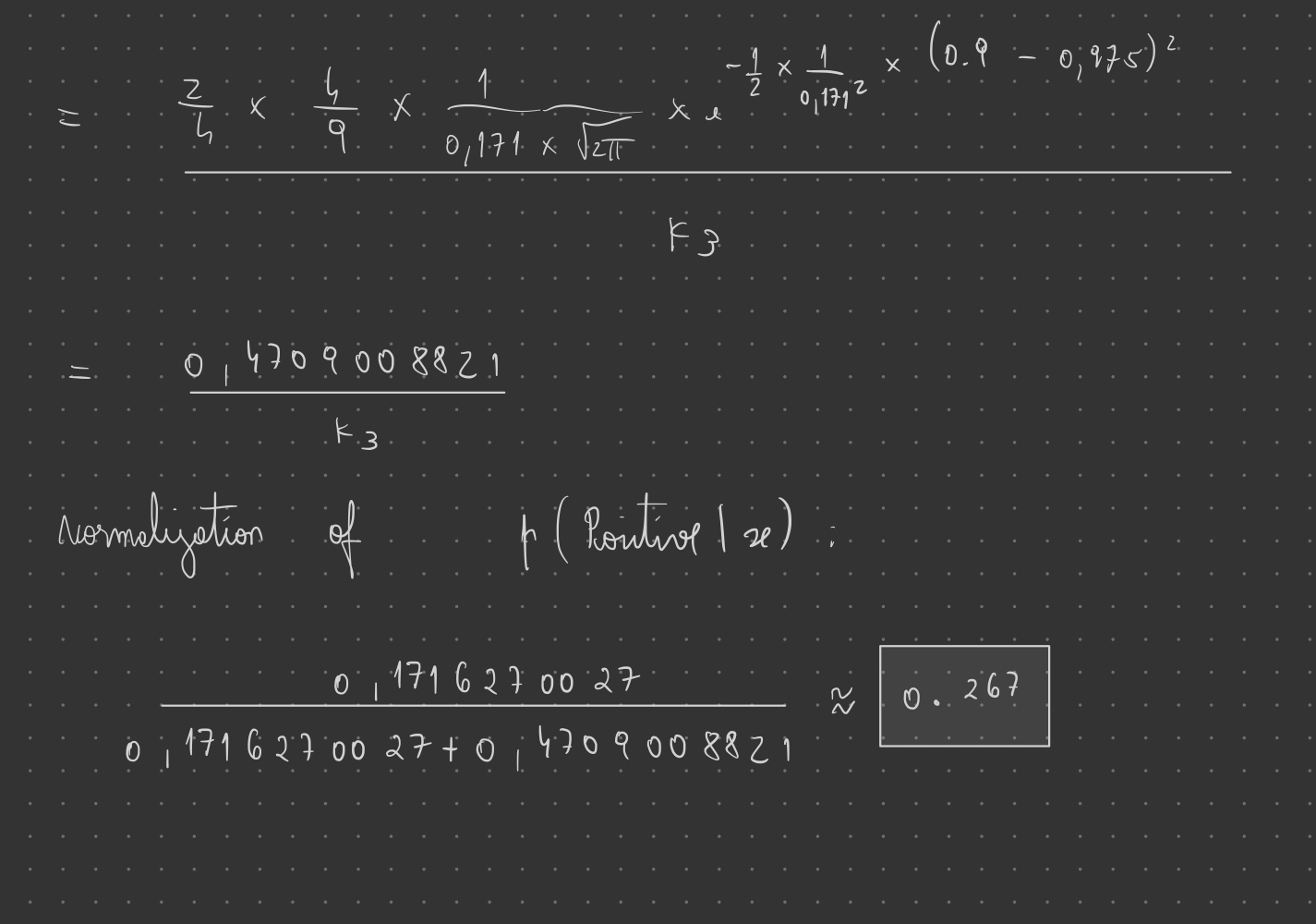
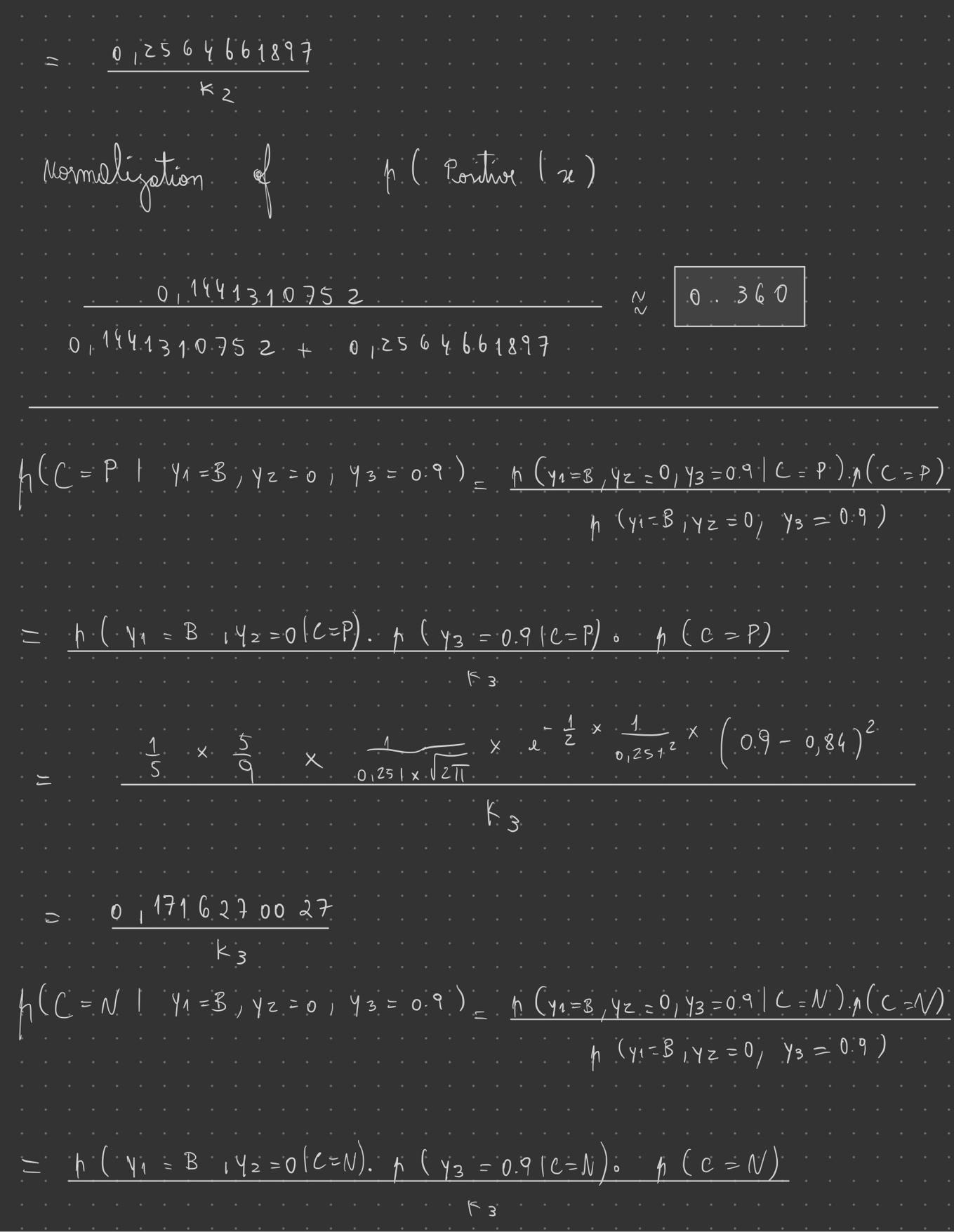
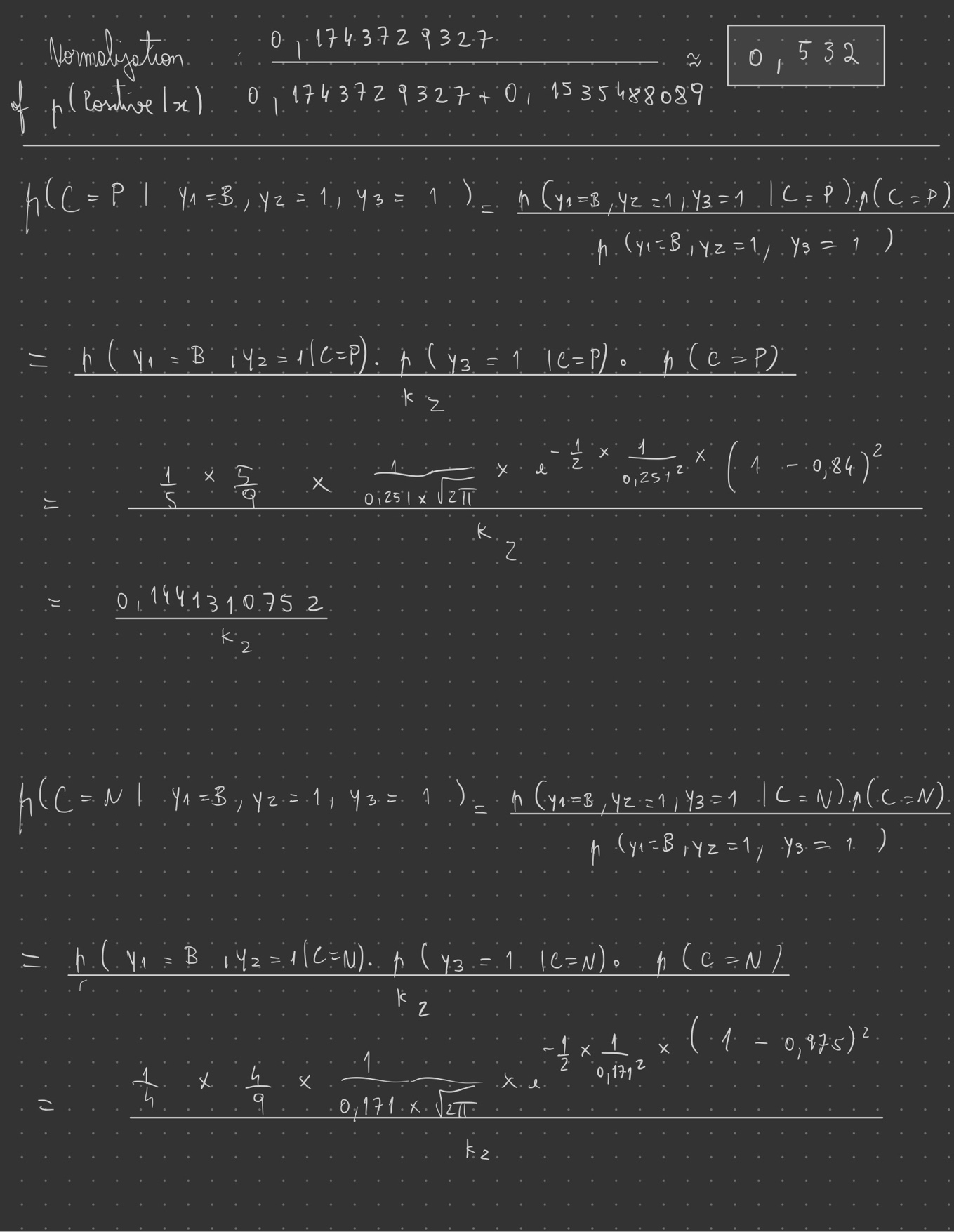


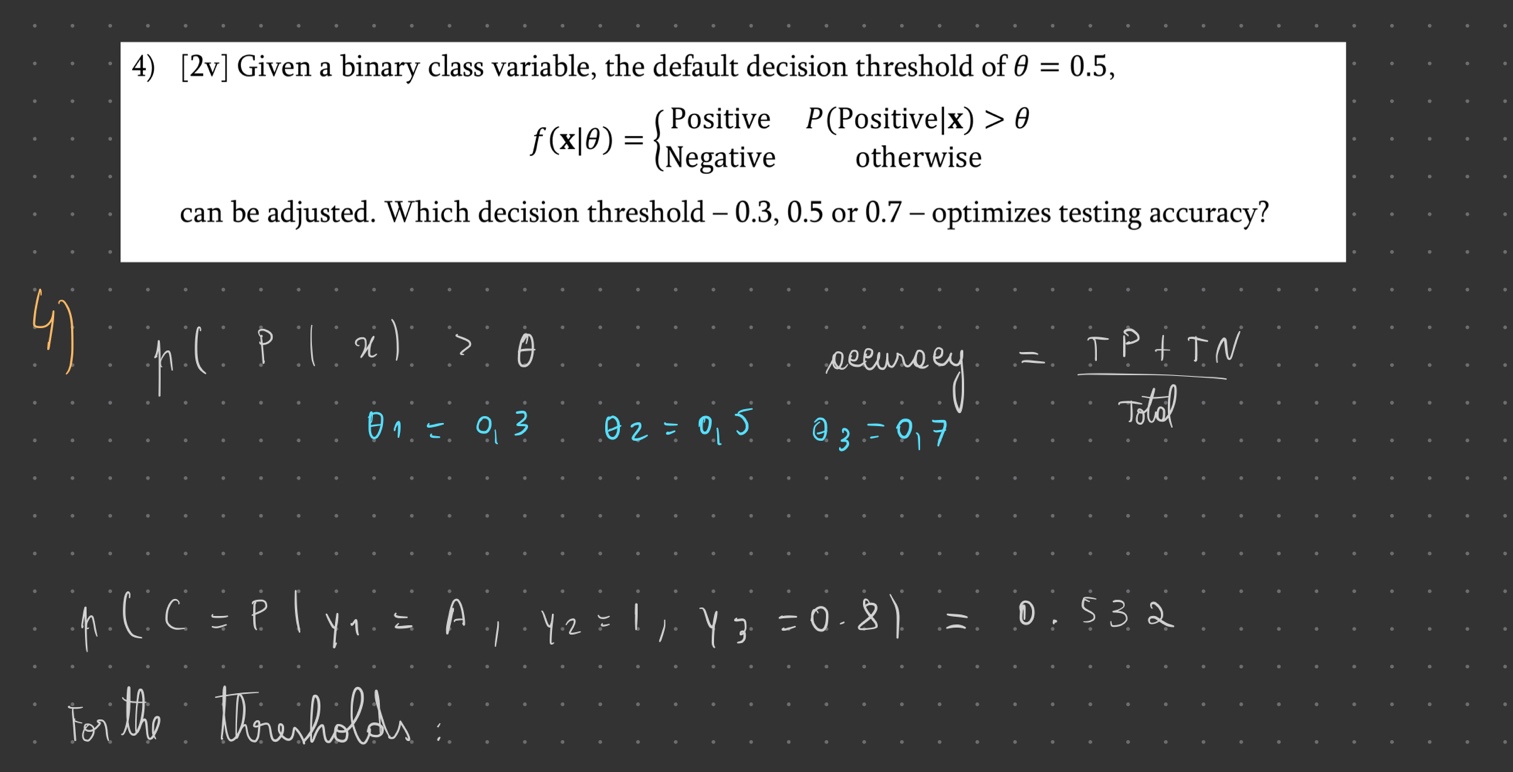


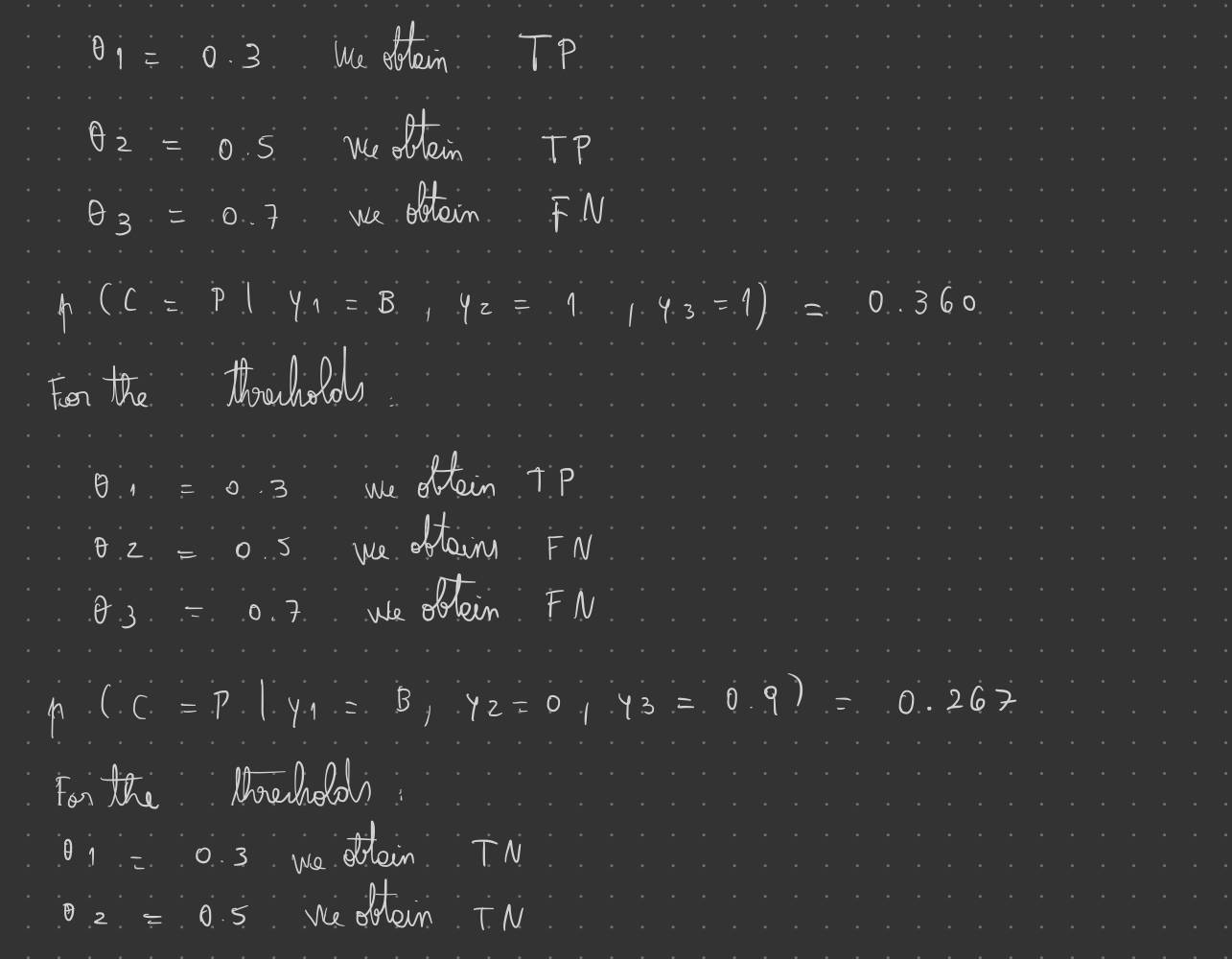
**2) Answer 2**

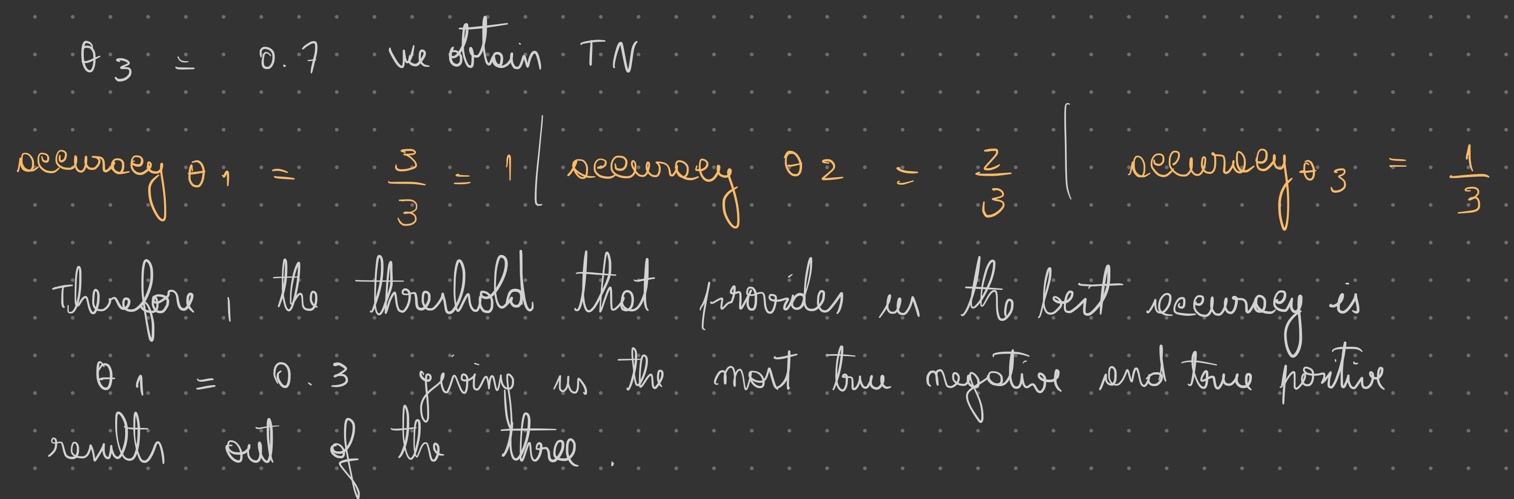


**3) Answer 3**



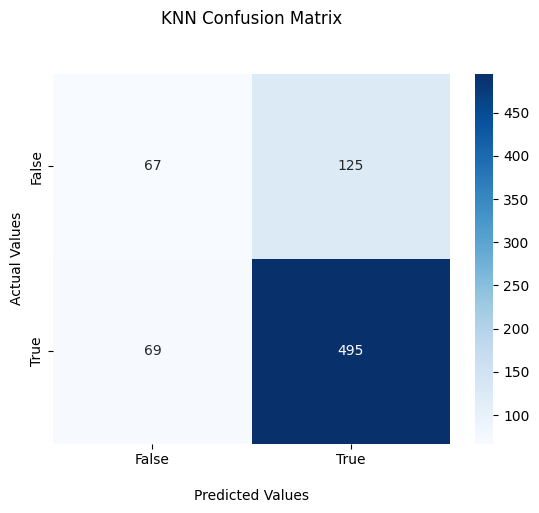
**4) Answer 4**

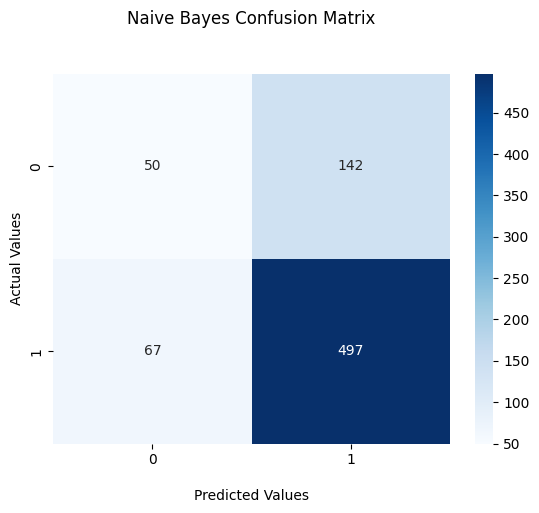
****

****

**II. Programming and critical analysis**

1. **Answer 1**





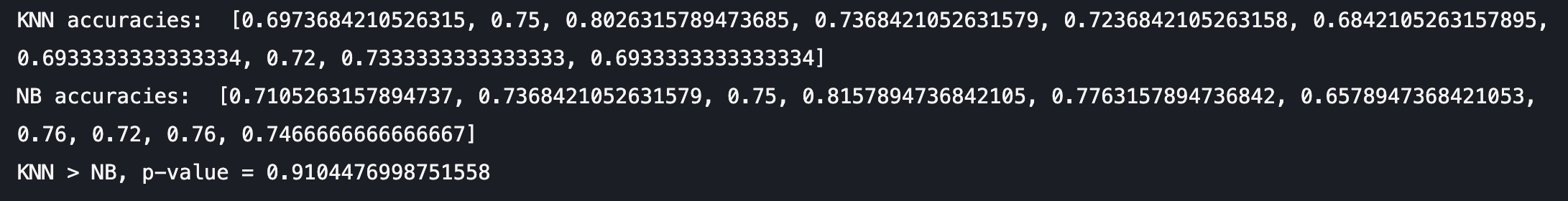
**2) Answer 2**

Assuming the following hypothesis:

- H0: “kNN is statistically similar to Naïve Bayes regarding accuracy”

- H1: “𝑘NN is statistically superior to Naïve Bayes regarding accuracy”

The obtained p-value reveals that we should reject H0 for significance levels approximately above 91.1%. The higher the p-value, the stronger evidence that the null hypothesis should not be rejected. Therefore, kNN is not statistically superior to Naïve Bayes regarding accuracy. Taking a look at the obtained accuracies for both kNN and Naïve Bayes, we can observe that the Naïve Bayes experiment displays higher overall accuracies which corroborates the obtained p-value.



**3) Answer 3**

Most of the time, Naïve Bayes is highly accurate when applied to large amounts of data. These are some reasons why Naïve Bayes could outperform kNN in terms of accuracy:

* kNN struggles the most when the number of inputs is very large. With the increase of dimensions comes an exponential increase in the volume of the input space. In high dimensions, points that appear to have many similarities, might be far away from each other.
* In addition, kNN could lead to overfitting when using small values for k as the algorithm gets too accustomed to the training data. On the other hand, high values for k could lead to underfitting where no good predictive patterns are found in the training data.
* Naïve Bayes, contrary to kNN, shines when solving multi-class prediction problems. If the independence assumption holds, a Naïve Bayes classifier performs better than most other models and less training data is required.
* Finally, Naïve Bayes works better when input variables are categorical, therefore, the more categorical variables, the more likely it is to outperform other models.

**III. APPENDIX**

**import** pandas **as** pd

**from** scipy.io.arff **import** loadarff

**from** sklearn.feature\_selection **import** mutual\_info\_classif, SelectKBest

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

**import** seaborn **as** sns

**from** sklearn **import** metrics, datasets

**from** sklearn.model\_selection **import** cross\_val\_score, StratifiedKFold

**from** sklearn.neighbors **import** KNeighborsClassifier

**from** sklearn.naive\_bayes **import** GaussianNB

**from** sklearn.metrics **import** classification\_report, confusion\_matrix

**from** scipy **import** stats

data **=** loadarff('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']

classNames **=** ['class']

accuraciesKNN **=** []

accuraciesNB **=** []

**def** **modelEvaluation**(**X**, **y**):

folds **=** StratifiedKFold(**n\_splits=10**, **random\_state** **=** **0**, **shuffle** **=** True)

KNN\_confMatrix **=** np.zeros((**2**, **2**))

NB\_confMatrix **=** np.zeros((**2**, **2**))

emptyMatrices **=** True

KNNPredictor **=** KNeighborsClassifier(**n\_neighbors** **=** **5**, **metric** **=** 'euclidean', **weights** **=** 'uniform')

NBPredictor **=** GaussianNB()

splitFolds **=** folds.split(X, y)

*# iterate per fold*

**for** train\_k, test\_k **in** splitFolds:

X\_train, X\_test **=** X.iloc[train\_k], X.iloc[test\_k]

y\_train, y\_test **=** y.iloc[train\_k], y.iloc[test\_k]

KNNPredictor.fit(X\_train, y\_train)

y\_KNNPred **=** KNNPredictor.predict(X\_test)

KNN\_auxConfMatrix **=** confusion\_matrix(y\_test, y\_KNNPred)

**if** emptyMatrices:

KNN\_confMatrix **=** KNN\_auxConfMatrix

**else**:

KNN\_confMatrix **+=** KNN\_auxConfMatrix

accuraciesKNN.append(metrics.accuracy\_score(y\_test, y\_KNNPred))

NBPredictor.fit(X\_train, y\_train)

y\_NBPred **=** NBPredictor.predict(X\_test)

NB\_auxConfMatrix **=** confusion\_matrix(y\_test, y\_NBPred)

**if** emptyMatrices:

NB\_confMatrix **=** NB\_auxConfMatrix

**else**:

NB\_confMatrix **+=** NB\_auxConfMatrix

emptyMatrices **=** False

accuraciesNB.append(metrics.accuracy\_score(y\_test, y\_NBPred))

**return** NB\_confMatrix, KNN\_confMatrix

**def** **plot\_confusion\_matrixes**(**KNN\_confMatrix**, **NB\_confMatrix**):

ax1 **=** sns.heatmap(KNN\_confMatrix, **annot** **=** True, **fmt** **=** "d", **cmap** **=** 'Blues')

ax1.set\_title('KNN Confusion Matrix\n\n');

ax1.set\_xlabel('\nPredicted Values')

ax1.set\_ylabel('Actual Values');

ax1.xaxis.set\_ticklabels(['False','True'])

ax1.yaxis.set\_ticklabels(['False','True'])

plt.show()

ax2 **=** sns.heatmap(NB\_confMatrix, **annot** **=** True, **fmt** **=** "d", **cmap** **=** 'Blues')

ax2.set\_title('Naive Bayes Confusion Matrix\n\n');

ax2.set\_xlabel('\nPredicted Values')

ax2.set\_ylabel('Actual Values');

plt.show()

**def** **testHypothesis**(**accuraciesKNN**, **accuraciesNB**):

**print**("KNN accuracies: ", accuraciesKNN)

**print**("NB accuracies: ", accuraciesNB)

res **=** stats.ttest\_rel(accuraciesKNN, accuraciesNB, **alternative** **=** "greater")

**print**("KNN > NB, p-value =", res.pvalue)

KNN\_confMatrix, NB\_confMatrix **=** modelEvaluation(X, y)

plot\_confusion\_matrixes(KNN\_confMatrix, NB\_confMatrix)

testHypothesis(accuraciesKNN, accuraciesNB)

**END**