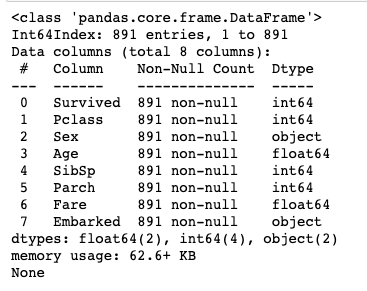
**Data acquisition**

The **FIRST** dataset I used in this chapter is the BayesAssign1\_02.csv data provided by Dr. Santago. This dataset has 1000 rows and 2 columns. The first column corresponds to the class of the data instance and the second column corresponds to its only continuous feature. There are two categories for the class variable: “POS” and “NEG”. On the other hand, the only feature is generated using two Gaussian distributions with different mean and variance.

The **SECOND** data I used is the Titanic data I downloaded from Kaggle and preprocessed (all missing data imputed). The dataset has 891 rows and 8 columns. Among them, the dataset has 7 features to predict whether or not a person survived on titanic. The categorical features are “Sex” and “Embarked”; while the continuous features are “Pclass”, “Age”, “SibSp”, “Parch”, and “Fare”. Here shown below is a summary of the dataset.



**Program development**

I developed two programs in this chapter.

For the **FIRST** program, I built a Naive Bayes Classifier on the BayesAssign1\_02.csv dataset to predict whether a data instance belongs to “POS” class or “NEG” class. The detailed codings can be found in the file “BayesAssign1\_PatrickFan.ipynb”.

1. The first thing I did is to separate my dataset into the train set and the test set. The constant TRAINING\_FRACTION (I used 0.7) is set to be the desired fraction of data I used (via random sampling) as train data; and the rest is used as test data.
2. Through dividing the train data into “POS” group and “NEG” group (according to their class label), I was able to calculate the mean and standard deviation values for each group. Since the two groups can be viewed as having the Gaussian distribution, I was able to find their probability distribution functions using the means and the standard deviations.
3. The next step is to predict the class labels for the test samples, given only their feature values. The posterior probability can be calculated using the formula P(y|x) = P(x|y) \* P(y) / P(x). Through comparing the posterior probability when y = “POS” and y = “NEG”, I was able to find the class label that will maximize the posterior probability, which I then assigned to this test instance as its predicted class label.

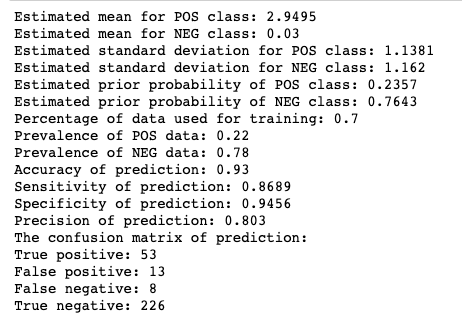
The **OTHER** program I wrote is about using the package KNeighborsClassifier from sklearn.neighbors to build a K nearest neighbor model to predict survival based on the fare one pays to get on the Titanic (the Titanic dataset). The coding can be found in the file “K Nearest Neighbor.ipynb”.

1. Once again, I separate my dataset into the train set and the test set. As a lazy learner, the train data of this program are stored in memory until the test instances are input for classification.
2. By setting the default parameters (the default number of nearest neighbors used is 5), I am able to build a K-Nearest Neighbor Classifier. The classifier is then used to predict the class value of the test instances.
3. Calculate some evaluation measures based on the results. The evaluation measures are then used to evaluate the effectiveness of the model.

**Data analysis and package use**

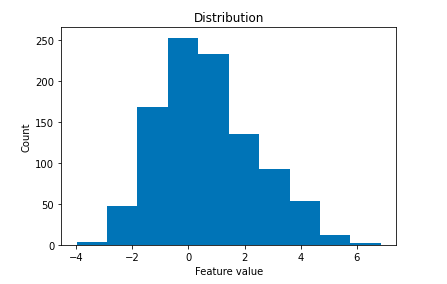
I use python in Jupyter Notebook as my programming language and IDE. The packages I used are math, random, sklearn, and matplotlib in python.

In the **FIRST** program, I did more than just building a Naive Bayes Classifier to predict the class a test instance belongs to. In order to explore the different evaluation measures, I also calculated the accuracy, sensitivity, specificity, precision, and the confusion matrix after I have the predicted class labels. I also interpret some of these measures in order to understand the data better. Here shown below is the output:

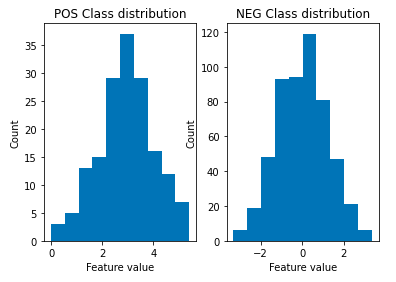


From the output, we can see that the model is showing good evaluation measurements: accuracy, sensitivity, specificity, and precision all have values above 0.8, showing that the model is showing good classification capability. Specifically, since the model has really high specificity, we should be confident in its ability to predict any negative instances as negative. Another aspect of the model is that it has high accuracy. As a result, we can say that most of the predictions given by the model are correct.

Another thing I did with the Naive Bayes Classifier is to plot the feature distribution before and after grouping the train data into “POS” and “NEG” groups. Here shown below is the feature distribution before grouping according to the class:



The next image is the feature distribution after the grouping.

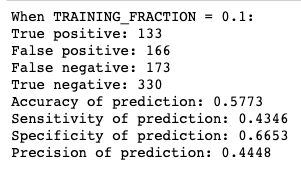


Even though the pattern is not transparent before the grouping, the patterns of Gaussian distribution become really clear after the grouping. Through visualizing the distributions of the two classes, we can be more confident in conducting the Naive Bayes Classification.

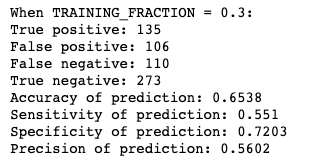
In my **SECOND** program, I first experimented with subdividing the raw dataset into the train and the test set using different training fractions--I wanted to explore how changing the training fraction influences the performance of the model. The four different training fraction numbers I used are 0.1, 0.3, 0.7, and 0.9. For each fraction value I used, I also output the corresponding evaluation measures so that I can compare the different models with each other.

Here below are the four output:

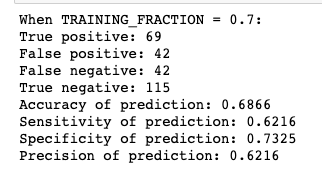
TRAINING\_FRACTION = 0.1:



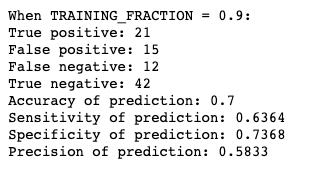
TRAINING\_FRACTION = 0.3:



TRAINING\_FRACTION = 0.7:

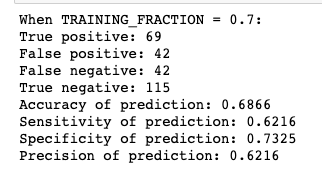


TRAINING\_FRACTION = 0.9:



1. We can see from the above outputs that comparatively the evaluation measures for low TRAINING\_FRACTION have lower values. In addition, we also see a steady growth in the resulting accuracy, sensitivity, and the other evaluation measurements as we change from using 0.1 as the TRAINING\_FRACTION to using 0.3, and then to using 0.7. The underlying mechanics behind all these is that: with a smaller fraction ratio, there are not enough samples to capture all information in the data, and therefore the results have lower accuracies, sensitivities, specificities and precisions.
2. When changing from using 70% of total instances as train data to using 90%, there is only little growth in accuracy, sensitivity, specificity; and there is even some decline in precision. I think this demonstrates that with 70% of total instances, almost all information that is contributive to the model building is already captured in the data. Thus, even though we are using more training instances to build the KNN model, the results have not improved much. Also, when using a large fraction of data instances as train data with little left as test data, the result may be misleading since the test data itself can be biased sometimes.

There is also another aspect that I experimented with for my dataset. I compared the majority voting approach with the distance-weighted voting approach when voting for how an instance should be classified. However, the results my dataset gives are the same for both approaches. Here shown below are results when I set the parameter weights as 'distance' (when using TRAINING\_FRACTION = 0.7):



Still, even though the results shown for my dataset are the same for the majority voting approach and the distance-weighted voting approach, there are cases when the two approaches give vastly different results. I am eager to explore more datasets when that is the result.

**Student learning summary and self-assessment**

Through the study of this chapter, I was able to learn more classification models beyond the basic concepts provided in chapter 3. Rule-based Classifier, Nearest Neighbor Classifier, and Naive Bayes Classifier. I think through working on the given dataset, I am now most familiar with the Naive Bayes Classifier, and I am confident in using Naive Bayes Classifiers to handle real-world data.

Another aspect of this chapter that I think is really interesting is how to evaluate the performance of a model. Among the various type of evaluation measures, I was able to experiment a lot with precision, sensitivity, specificity, etc., and how they may relate to each other.

Something I want to discover more:

1. I want to explore the ROC, AUC, PR curve (the aggregate evaluation) more using real-world data like the Titanic dataset I used for chapter 3. I also want to explore the ways they are either sensitive or not sensitive to skewness.
2. I want to build my own rule-based classifier and KNN classifiers in order to better understand their underlying principles.

Self-evaluation:

I believe that my understanding of notions related to the Naive Bayes Classifier, the K-Nearest Neighbors, and the evaluation measures is solid. I am confident in applying those concepts to real world problems, and fit in a good set of parameters to adjust for the best performance of a model. Based upon those understanding, I would like to give myself an A on this chapter.