Introduction

In this assignment I have created a generic Naives Bayes binary classification algorithm that allows the user to pass in classes and a data file of their choice to conduct the algorithm, performing repeated K-Folds and ultimately producing an overall mean accuracy score. (Brownlee, 2016)

How to run the program

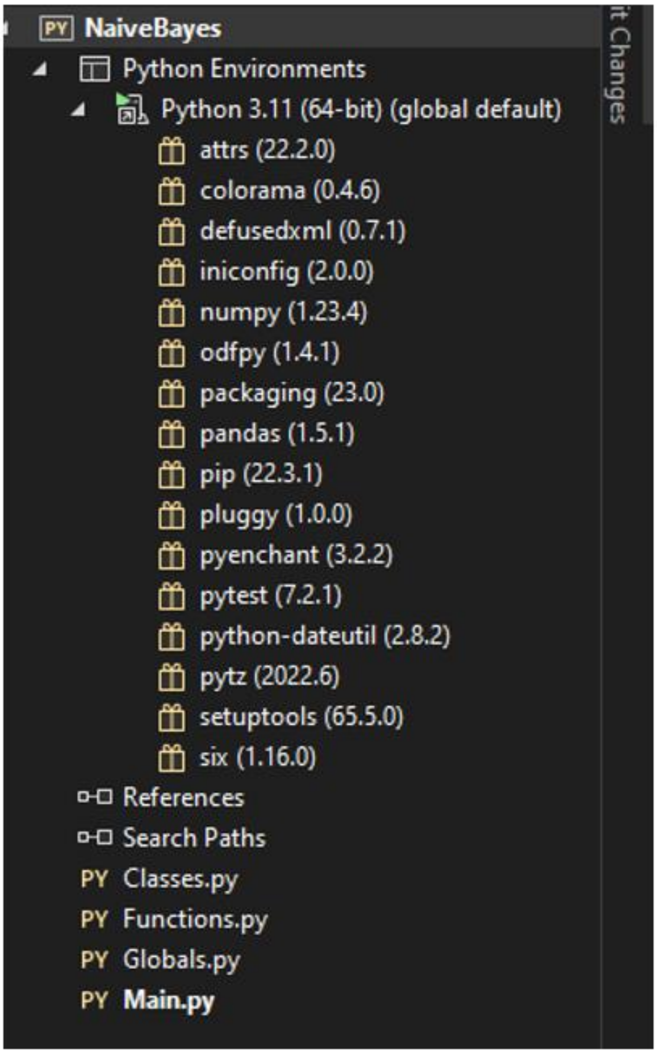
- The program is written Python and should run with Python version 3.11.
- Upon creation, this project was built and executed in Visual Studio 2022.
- Packages needed should just be the pandas package. However, if unsuccessful, please see the package list below in Fig.2.
- Upon executing the file the user will be prompted to for 3 user input fields. These fields are…
    - Class A value → the default value is 'Win' class for the default file.
    - Class B value → the default value is for the 'Loss' class for the default file.
    - The file path of your ods data file → default './StartingXI_WinLoss.ods'

Fig.1 User Input entry example with values that work.

```
C:\Users\Willi\AppData\Local\Programs\Python\Python311\python.exe
Enter class A value (Default: Win): Win
Enter class B value (Default: Loss): Loss
Enter data file path (Default: ./StartingXI_WinLoss.ods) : ./StartingXI_WinLoss.ods
```

When prompted the user can press enter 3 times to take the default values for all 3 inputs, and successfully run the implementation with the data file provided. The purpose for this method was to allow for greater user freedom to use their own classes and data files, and to make this implementation more generic.

Fig.2 Package And Module List.



<u>Program data</u>

- The dataset was created by myself for the purpose of this assignment and the dataset can be seen in Fig.2. Accessed in the project here ‘./NaiveBayes/NaiveBayes/StartingXI_WinLoss.ods’.
- The attributes in the data file are positions in a football team. The 12 attributes are ‘Result’, 'GoalKeeper', 'Right Center Back', 'Right Back', 'Left Back', 'Left Center Back', 'Defensive Midfielder', 'Attacking Midfielder', 'Central Midfielder', 'Center Forward', 'Left Winger', 'Right Winger'. However, attributes are not hardcoded, and attributes can be parsed into the program when the user enters a new dataset, potentially with a greater/lesser number of attributes.

- Attribute types in the ‘Result’ attribute are binary, i.e ‘Win’ or ‘Loss’. The other 11 attributes have values that are nominal/categorical, e.g. Goalkeeper has 3 values with no order. The data types of the attribute and values are strings.

- Data is discrete, i.e. can be counted.

- The dataset is stored in an excel open document spreadsheet that **contains 30 records**.

Fig.3 Dataset used.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Result	GoalKeeper	Right Center Back	Right Back	Left Back	Left Center Back	Defensive Midfielder	Attacking Midfielder	Central Midfielder	Center Forward	Left Winger	Right Winge
2	Win	Allison	Gomez	Alexander Arnol	Robertson	Virgil Van Dijk	Fabhino	Jones	Arthur	Firmino	Diaz	Salah
3	Win	Kelleher	Matip	Bajectic	Robertson	Virgil Van Dijk	Fabhino	Keita	Thiago	Diaz	Salah	Firmino
4	Win	Allison	Virgil Van Dijk	Alexander Arnol	Tsimikas	Gomez	Arthur	Carvalho	Jones	Salah	Jota	Nunez
5	Loss	Kelleher	Kouyate	Chambers	Milner	Matip	Arthur	Jones	Thiago	Nunez	Salah	Firmino
6	Loss	Kelleher	Matip	Bajectic	Tsimikas	Kouyate	Fabhino	Carvalho	Oxlade-Chamberlain	Jota	Nunez	Diaz
7	Loss	Adrian	Kouyate	Bajectic	Milner	Gomez	Oxlade-Chamberlain	Keita	Jones	Nunez	Jota	Firmino
8	Win	Allison	Matip	Chambers	Tsimikas	Kouyate	Thiago	Henderson	Clarke	Salah	Nunez	Diaz
9	Loss	Adrian	Virgil Van Dijk	Chambers	Milner	Gomez	Fabhino	Arthur	Elliot	Firmino	Jota	Salah
10	Win	Allison	Kouyate	Alexander Arnol	Milner	Virgil Van Dijk	Arthur	Henderson	Thiago	Diaz	Salah	Firmino
11	Win	Kelleher	Gomez	Ramsey	Robertson	Matip	Thiago	Carvalho	Oxlade-Chamberlain	Salah	Nunez	Jota
12	Loss	Allison	Kouyate	Bajectic	Tsimikas	Gomez	Arthur	Jones	Clarke	Firmino	Nunez	Jota
13	Loss	Adrian	Virgil Van Dijk	Ramsey	Milner	Kouyate	Arthur	Elliot	Henderson	Salah	Jota	Nunez
14	Win	Adrian	Kouyate	Alexander Arnol	Tsimikas	Matip	Thiago	Arthur	Henderson	Diaz	Salah	Firmino
15	Win	Allison	Virgil Van Dijk	Chambers	Milner	Matip	Fabhino	Keita	Oxlade-Chamberlain	Diaz	Salah	Jota
16	Loss	Adrian	Gomez	Alexander Arnol	Tsimikas	Matip	Arthur	Keita	Carvalho	Diaz	Salah	Firmino
17	Win	Kelleher	Gomez	Ramsey	Robertson	Kouyate	Oxlade-Chamberlain	Jones	Arthur	Firmino	Diaz	Salah
18	Win	Allison	Gomez	Ramsey	Robertson	Matip	Thiago	Carvalho	Keita	Diaz	Nunez	Jota
19	Win	Adrian	Virgil Van Dijk	Ramsey	Tsimikas	Kouyate	Thiago	Henderson	Jones	Firmino	Nunez	Diaz
20	Loss	Adrian	Virgil Van Dijk	Chambers	Robertson	Kouyate	Arthur	Carvalho	Keita	Nunez	Diaz	Firmino
21	Win	Kelleher	Virgil Van Dijk	Alexander Arnol	Robertson	Kouyate	Oxlade-Chamberlain	Jones	Henderson	Jota	Nunez	Salah
22	Loss	Adrian	Gomez	Chambers	Milner	Virgil Van Dijk	Fabhino	Carvalho	Oxlade-Chamberlain	Firmino	Diaz	Jota
23	Win	Adrian	Matip	Alexander Arnol	Milner	Virgil Van Dijk	Arthur	Henderson	Clarke	Jota	Diaz	Firmino
24	Loss	Adrian	Kouyate	Chambers	Robertson	Virgil Van Dijk	Arthur	Henderson	Oxlade-Chamberlain	Firmino	Jota	Salah
25	Win	Adrian	Gomez	Ramsey	Robertson	Virgil Van Dijk	Fabhino	Carvalho	Henderson	Nunez	Jota	Diaz
26	Win	Kelleher	Kouyate	Ramsey	Tsimikas	Matip	Arthur	Keita	Oxlade-Chamberlain	Nunez	Jota	Salah
27	Loss	Kelleher	Matip	Alexander Arnol	Robertson	Kouyate	Oxlade-Chamberlain	Arthur	Clarke	Salah	Nunez	Firmino
28	Loss	Kelleher	Matip	Ramsey	Tsimikas	Kouyate	Arthur	Elliot	Henderson	Firmino	Diaz	Jota
29	Win	Allison	Virgil Van Dijk	Alexander Arnol	Tsimikas	Matip	Fabhino	Keita	Thiago	Nunez	Salah	Jota
30	Win	Allison	Virgil Van Dijk	Alexander Arnol	Milner	Matip	Fabhino	Keita	Elliot	Firmino	Jota	Salah
31	Win	Allison	Virgil Van Dijk	Alexander Arnol	Robertson	Kouyate	Oxlade-Chamberlain	Carvalho	Thiago	Firmino	Diaz	Salah

Pre-processing

Whilst the implementation above takes in a default dataset that is already cleaned and in the correct format to use the application, to use this algorithm the data should be presented and accessible as such…

- The Data file will need to be in '.ods' format. The file must be in a location accessible to the program as the file path must be entered on starting of the program.
- The data can have any number of attributes.
- The dataset's first attribute must contain your two classes as values, for example the 'Result' attribute in fig.2 contains the two classes as values ('Win'/'Loss'). The name of the attribute does not matter, but the attribute values do. The classes you choose will be what you enter on execution of the program via the user input.
- The other attributes can have multiple categorical/nominal values as in a string data type.
- All data must be discrete.
- Data needs to be cleaned. How:
    - Only two classes represented of your choice, e.g., 'Win' & 'Loss'.
    - Analysis should take place to correct spelling mistakes, missing values with appropriate values for that attribute. This will remove noise.
    - If numerical data is used, methods such as binning could be used to clean and optimise the data(*A Simple Guide to Data Preprocessing in Machine Learning*,n.d.).

  Note: Smoothing of the data takes place in the code using 'Laplace smoothing, using an alpha for the count of each attribute value(Jayaswal, 2020).

Naives Bayes(Classifier)

Definition: Naives bayes is a classification algorithm, or supervised machine learning algorithm, and is used to sort object based on certain classes, groups or characteristics. It assumes that classes are conditionally independent, meaning the information does not interact, when forming its predictions(*What Is Naïve Bayes | IBM*, n.d.).

See appendix for algorithm choice and usage.

Formal Description:

Fig.4 Formal Equation of Naives Bayes algorithm

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

(Source: https://en.wikipedia.org/wiki/Naive_Bayes_classifier)

Fig.6 Plain English Equation translation

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

(Source: https://en.wikipedia.org/wiki/Naive_Bayes_classifier)

The formula states in Fig.4…

- $\mathbf{C_k}$ is the class variable and **x** is the dependent feature.
- P($\mathbf{C_k}$ | **x**) is the posterior probability, i.e is the probability of the event after evidence is seen. This is the probability we wish to calculate and form our prediction on.
- P(**x** | $\mathbf{C_k}$) is the probability of $\mathbf{C_k}$ occurring given that the evidence **x** is true. This is known as the likelihood.
- P($\mathbf{C_k}$) is the prior probability, i.e the probability of $\mathbf{C_k}$ occurring based on no evidence, but the prediction of the result/outcome.
- P(**x**) is the evidence, and this value does not change as it has already occurred.

  (Kumar, 2019)

Likewise, the Naives Bayes algorithm states that we are given a feature vector of size n, seen in Fig.5. Each dependent feature of the set is applied to the algorithm in Fig.4.

*Fig.5 Dependent features **X** of class* $\mathbf{C_k}$

$$p(C_k, x_1, \ldots, x_n)$$

(Source: https://en.wikipedia.org/wiki/Naive_Bayes_classifier)

Informal description

The algorithm operates on these steps:

1. Calculate the prior/class probabilities for each item relative to the class we are predicting. This will be the class frequency divided by the sum of both class frequencies. This is the base prediction/hypothesis on the training data. This can be seen in Fig.7.

   Fig.7 code calculation of prior probabilities

```
ClassAPriorProb = (ClassATrainingData.ClassFrequency  / (ClassATrainingData.ClassFrequency  +  ClassBTrainingData.ClassFrequency))
ClassBPriorProb = (ClassBTrainingData.ClassFrequency  /  (ClassATrainingData.ClassFrequency  +  ClassBTrainingData.ClassFrequency))
```

2. Calculate the conditional probabilities for each dependent feature and multiply each all these together.
3. Multiply the prior probability with sum of the conditional probabilities to form posterior probability.

   The code in Fig.8 shows steps 2 & 3.

Fig.8 Calculating the posterior probability from the conditional and class probabilities.

```
for index, testDataRow in TestDataDF.iterrows():
    ClassAPostProb  = ClassAPriorProb  #Store Prior Probability P(ClassA)
    ClassBPostProb = ClassBPriorProb   #Store Prior Probability P(ClassB)
    for attribute in Globals.attributes[1:len(Globals.attributes)]: #skip class column, only dependent feature attributes
        ClassAPostProb  *= ((ClassATrainingData.Attribute_Values_Dict.get(testDataRow[attribute]) / ClassATrainingData.ClassFrequency))
        ClassBPostProb  *= ((ClassBTrainingData.Attribute_Values_Dict.get(testDataRow[attribute]) / ClassBTrainingData.ClassFrequency))
```

4. The raw probabilities now need to be normalized to get the class probabilities.

   This is done by dividing the class A posterior probability by the sum of class A & B posterior probabilities. This is seen in Fig.9.

Fig.9 Normalising the posterior probabilities.

```
ClassAProb = (ClassAPostProb / (ClassAPostProb + ClassBPostProb)) #Normalisation of probabilities
ClassBProb = (ClassBPostProb / (ClassAPostProb + ClassBPostProb))
```

5. The last step is to contrast the two normalized probabilities to predict the class outcome based on the Naives bayes algorithm. This is seen in Fig.8.
   - If the probability for class A is greater than the class B, Naives predicts class A will occur.
   - If the probability for class B is greater than the class A, Naives predicts class B will occur.
   - If the probabilities are equal, the outcome is “Undetermined” by the algorithm.

Fig.10 Forming a prediction on the class probabilities (“Undetermined” if probabilities equal)

```
result = Globals.classA if (ClassAProb > ClassBProb) else Globals.classB if (ClassAProb < ClassBProb) else "Undetermined"
predictedResults.append(result) #store all results in array
```

Metrics used to evaluate the performance.

In terms of performance, we used the resampling technique k-fold cross validation, repeatedly, with numerous folds from the range of 3 to 10 folds. Reasoning for this is it would produce ‘a more reliable estimate of model performance than the result of a single k-fold cross-validation procedure’(Brownlee, 2020).

The mean accuracy score we ascertained for correct predictions, performed on the 30 entries in our dataset, using folds 3, 5, 6, 10, was 74.167%(3dp). Likewise, a confusion matrix is displayed for each respective fold for correct and incorrect predictions(Suresh, 2020)(Brownlee, 2013). These results are seen in Fig.10.

Fig.11 Confusion matrices for each K-fold used and mean calculation for the respective K-fold

```
Enter class A value (Default: Win):
Enter class B value (Default: Loss):
Enter data file path (Default: ./StartingXI_WinLoss.ods) :

                Confusion Matrix - 3 folds
                      Win  |      Loss
Predicted Win  |      14   |       2
Predicted Loss |      4    |       10

This means that out of 30 test inputs naives predicted 24 correctly and with  6 incorrect predicitions.
This gives us a mean accuracy score for our NaivesBayes implementation of 80.0 percent.

                Confusion Matrix - 5 folds
                      Win  |      Loss
Predicted Win  |      13   |       3
Predicted Loss |      5    |       9

This means that out of 30 test inputs naives predicted 22 correctly and with  8 incorrect predicitions.
This gives us a mean accuracy score for our NaivesBayes implementation of 73.33333333333333 percent.

                Confusion Matrix - 6 folds
                      Win  |      Loss
Predicted Win  |      13   |       4
Predicted Loss |      5    |       8

This means that out of 30 test inputs naives predicted 21 correctly and with  9 incorrect predicitions.
This gives us a mean accuracy score for our NaivesBayes implementation of 70.0 percent.

                Confusion Matrix - 10 folds
                      Win  |      Loss
Predicted Win  |      14   |       4
Predicted Loss |      4    |       8

This means that out of 30 test inputs naives predicted 22 correctly and with  8 incorrect predicitions.
This gives us a mean accuracy score for our NaivesBayes implementation of 73.33333333333333 percent.

Overall mean accuracy score across multiple K-Folds is... 74.16666666666667
Press any key to continue . . .
```

Performance discussion

The algorithm performed well in terms of accuracy with a mean of 74.167%. This did deviate but not significantly with the minimum and max accuracy of 70% and 80% respectively. However, this could be due to a small dataset utilised and at certain k-fold used will underfit or overfit the data. Conversely, I speculate the time complexity would be heavily impacted if the dataset was larger(e.g.,1000 entries) given the numerous folds and smoothing of the data conducted.

Likewise, the algorithm assumes features are independent, when, if a certain player is playing this could influence the chemistry and performance of the team and thus the result. Meaning if this assumption does not hold true and can influence our accuracy to predict outcomes correctly using this model.

Lastly, performance could be influenced by anomalies/noise in the data, mitigating our ability to produce accurate predictions.

Programming Code

Main.py

```
import copy
import pandas as pd
import Globals
from Functions import NaiveBayes,  KFold, PrintConfusionMatrix, LaplaceSmoothing, TrainData,
UpdateClassDictValues

def main ():
   Globals.classA        = input("Enter class A value (Default: Win): ").replace(" ","") or "Win"   #Take user
class A value or use default value "Win" - replace used ot minimise error
   Globals.classB        = input("Enter class B value (Default: Loss): ").replace(" ","") or "Loss" #Take user
class B value or use default value "Loss"
   fileName            = input("Enter data file path (Default: ./StartingXI_WinLoss.ods) : ").replace(" ","") or
"./StartingXI_WinLoss.ods"
   try:
      DataDF              =  pd.read_excel(fileName) #Stores team data in a data frame
      DatasetNumRows        =  len(DataDF.index)   #Used to dedeuce folds to try & split quantity
      Globals.attributes     =  DataDF.columns      #Used in multiple functions
      CumulativeAccuracyScores =  0                #Track the accuracy scores across different folds
      PrePopulatedDataFreqObj  =  LaplaceSmoothing(DataDF) #populate object with alpha values
      K_FoldsUsed           =  0 #counts the number of folds we have completed
      for K_Folds in range(3,11): #try folds 3..10
         if not((DatasetNumRows % K_Folds) == 0): continue #skip fold  if the dataset cannot produce a
rational dataset split

         ClassResultsDict = { #Store predicition accuracies - resets on each new fold
                  "Predicted-ClassA&ActualClassA"   : 0, #Correct predicition for class A
                  "Predicted-ClassA&ActualClassB"   : 0, #Wrong predicition for class A
                  "Predicted-ClassB&ActualClassA"   : 0, #Wrong predicition for class B
                  "Predicted-ClassB&ActualClassB"   : 0  #Correct predicition for class B
              }
         splitQuantity  = int(DatasetNumRows / K_Folds) # Segment size for training and test data
         testSplitRange = [0, splitQuantity]         #Test data slice - used for test set/fold range

         for iteration in range(0, K_Folds):         #iterate through each fold
            NaivesDatasets = KFold(test_split_range = testSplitRange, dataDF = DataDF) #produces object
with training and test data
            ClassADataObj = copy.deepcopy(PrePopulatedDataFreqObj)              #assign a copy of the
object with prepopulated values - resets object each iteration
            ClassBDataObj = copy.deepcopy(PrePopulatedDataFreqObj)
            TrainData(dataframe = NaivesDatasets.Training, dataStoreObj = ClassADataObj, desiredClass =
Globals.classA) #Populate Class A training data Obj
            TrainData(dataframe = NaivesDatasets.Training, dataStoreObj = ClassBDataObj, desiredClass =
Globals.classB) #Populate Class B training data Obj
            ClassPredictedResults = NaiveBayes(NaivesDatasets.Test, ClassADataObj, ClassBDataObj)
#performs naive bayes returns predictions for that test data in that fold
            UpdateClassDictValues(TestDataDF=NaivesDatasets.Test, classResults = ClassPredictedResults,
ClassResultsDict = ClassResultsDict) #update stored class predicitions
            testSplitRange[0]+= splitQuantity
            testSplitRange[1]+= splitQuantity

         PrintConfusionMatrix(ClassResultsDict, K_Folds)
         CorrectPredictions   = (ClassResultsDict["Predicted-ClassA&ActualClassA"]  +
ClassResultsDict["Predicted-ClassB&ActualClassB"])
```

```
        IncorrectPredictions = (ClassResultsDict["Predicted-ClassB&ActualClassA"]  +
ClassResultsDict["Predicted-ClassA&ActualClassB" ])
        try:
          AccuracyScore = (CorrectPredictions / (IncorrectPredictions + CorrectPredictions)) #Used to
print out to console mean accuracy score
          CumulativeAccuracyScores+=AccuracyScore #stores all accuracy stores
          print("\nThis means that out of",(splitQuantity  * K_Folds), "test inputs naives predicted",
CorrectPredictions, "correctly and with ", IncorrectPredictions, "incorrect predicitions.")
          print("This gives us a mean accuracy score for our NaivesBayes implementation of",
(AccuracyScore*100),"percent.")
          K_FoldsUsed+=1
        except ZeroDivisionError:
          print("No mean produced as correct predicitions is",CorrectPredictions, "and incorrect
predicitons is",IncorrectPredictions,"resulting in no Mean value produced.")
      try:
        print("\nOverall mean accuracy score across multiple K-Folds is...",
((CumulativeAccuracyScores)/K_FoldsUsed)*100) #Print mean accuracy scores across folds
      except ZeroDivisionError:
        print("The mean cumulative accuracy score",(CumulativeAccuracyScores),"cannot be divided by
the number of kfolds used",K_FoldsUsed)
  except:
    print("Incorrect file name entered.")
    print("Please restart the program and try again.")


main() #Program entry
```

Functions.py

```
import pandas as pd
from Classes import Dataset, DataFrequencyStore
import Globals

def KFold(test_split_range, dataDF): #Splits thte dataframe into folds for training and test data
  Data = Dataset()
  testRangeStart = test_split_range[0]
  testRangeEnd = test_split_range[1]
  Data.Test = dataDF.iloc[testRangeStart: testRangeEnd] #Test data
  Data.Training = dataDF.iloc[0 : testRangeStart] #Training Data (first part)
  Data.Training  = pd.concat([Data.Training, dataDF.iloc[testRangeEnd : len(dataDF.index)]], axis=0)#
add Training data (second part) to first part
  return Data #returns object containing test and training data


def TrainData(dataframe, dataStoreObj, desiredClass):  #
  for index, row in dataframe.iterrows(): #will need to influence the numebr of rows
    if row[Globals.attributes[0]] == desiredClass: #result we care about(classA or classB)
      #Iterates through each column for that row
      for attribute in Globals.attributes[1:len(Globals.attributes)]: #start at 1 to avoid class value
          dataStoreObj.Attribute_Values_Dict[row[attribute]] +=1   #Each line is adding/updating the ( :
win count)
      dataStoreObj.ClassFrequency +=1 #Count frequency of class value

def NaiveBayes(TestDataDF, ClassATrainingData, ClassBTrainingData):
  predictedResults = []
  ClassAPriorProb = (ClassATrainingData.ClassFrequency  / (ClassATrainingData.ClassFrequency   +
ClassBTrainingData.ClassFrequency)) #Prior Probability of P(ClassA)
  ClassBPriorProb = (ClassBTrainingData.ClassFrequency  /  (ClassATrainingData.ClassFrequency  +
ClassBTrainingData.ClassFrequency)) #Prior Probability we lose P(ClassB)
```

```
  for index, testDataRow in TestDataDF.iterrows():
    ClassAPostProb  = ClassAPriorProb  #Store Prior Probability P(ClassA)
    ClassBPostProb = ClassBPriorProb   #Store Prior Probability P(ClassB)
    for attribute in Globals.attributes[1:len(Globals.attributes)]: #skip class column, only dependent
feature attributes
      ClassAPostProb  *= ((ClassATrainingData.Attribute_Values_Dict.get(testDataRow[attribute]) /
ClassATrainingData.ClassFrequency))  #Multiply each conditional probability for a win
      ClassBPostProb  *= ((ClassBTrainingData.Attribute_Values_Dict.get(testDataRow[attribute]) /
ClassBTrainingData.ClassFrequency))  #Multiply each conditional probability for a loss
    ClassAProb = (ClassAPostProb / (ClassAPostProb + ClassBPostProb)) #Normalisation of probabilities
    ClassBProb = (ClassBPostProb / (ClassAPostProb + ClassBPostProb))
    result = Globals.classA if (ClassAProb > ClassBProb) else Globals.classB if (ClassAProb < ClassBProb)
else "Undetermined"# determine outcome from highest probability, undetermined if equal
    predictedResults.append(result) #store all results in array
  return predictedResults #return array of results for this fold



def PrintConfusionMatrix(PredictedToActualResultsDict, fold): #add class variables - gloabl optimal
    print('\n           Confusion Matrix -',fold, 'folds\n                ',  Globals.classA,' |    ',Globals.classB )
    for key, value in PredictedToActualResultsDict.items():
      match key:
        case ('Predicted-ClassA&ActualClassA'):
          print('Predicted',  Globals.classA, ' |     ', value,'  |', end='')

        case ("Predicted-ClassB&ActualClassA"):
          print('Predicted',  Globals.classB, '|     ', value,'   |',end='')
        case _:
          print("     ", value)

def LaplaceSmoothing(dataframe):
  #Purpose is to populate the data with an alpha value --> avoid zero frequency problem/clean data
  #Jayaswal, V. (2020, November 22). Laplace smoothing in Naïve Bayes algorithm. Medium.
https://towardsdatascience.com/laplace-smoothing-in-na%C3%AFve-bayes-algorithm-9c237a8bdece
  alpha = 1 #value to prepopulate the data with
  PrepopulatedDataObj =  DataFrequencyStore() #will store depenedent features with alpha value
  PrepopulatedDataObj.ClassFrequency = alpha #class number(1) * alpha
  for index, row in dataframe.iterrows():
    #iterate through each row value
    for value in row[1:len(row)]:
      if not value in PrepopulatedDataObj.Attribute_Values_Dict: # Not in the dictionary?
        PrepopulatedDataObj.Attribute_Values_Dict[value] = alpha #add data with alpha value
  return PrepopulatedDataObj #return populated data object

def UpdateClassDictValues(TestDataDF, classResults , ClassResultsDict):
  for index in range(0, len(classResults)): #iterate through results array
    rowClassValue = TestDataDF.iloc[index][0] #reduce lookups of class value in the row
    result   = classResults[index] #result naives predicted
    ClassResultsDict["Predicted-ClassA&ActualClassA"] +=  1 if(result  == Globals.classA  and
classResults[index] == rowClassValue )   else 0
    ClassResultsDict["Predicted-ClassB&ActualClassB"] +=  1 if(result  == Globals.classB  and
classResults[index] == rowClassValue )   else 0
    ClassResultsDict["Predicted-ClassA&ActualClassB"] +=  1 if(result  == Globals.classA  and
classResults[index] != rowClassValue )   else 0
    ClassResultsDict["Predicted-ClassB&ActualClassA"] +=  1 if(result  == Globals.classB  and
classResults[index] != rowClassValue )   else 0
```

Classes.py

```
import pandas as pd
class Dataset:
   def __init__(self):
      self.Training = pd.DataFrame() #stores training data
      self.Test     = pd.DataFrame() #stores test data

class DataFrequencyStore:  #used to group together information
    def __init__(self):
       self.ClassFrequency  = 0
       self.Attribute_Values_Dict  = {}
```

Globals.py

```
attributes    =  [] #global attributes
classA        = ""
classB        = ""
```

References

- Kumar, N. (2019, January 14). Naive Bayes Classifiers - GeeksforGeeks. GeeksforGeeks. https://www.geeksforgeeks.org/naive-bayes-classifiers/Starmer, Josh, director. Naive Bayes, Clearly Explained!!! YouTube, YouTube, 3 June 2020, https://www.youtube.com/watch?v=O2L2Uv9pdDA. Accessed 5 Oct. 2022.

- Brownlee, J. (2016, September 22). Naive Bayes for Machine Learning. Machine Learning Mastery. https://machinelearningmastery.com/naive-bayes-for-machine-learning/

- S, Y. (2020, May 8). An Introduction to Naïve Bayes Classifier. Medium. https://towardsdatascience.com/introduction-to-na%C3%AFve-bayes-classifier-fa59e3e24aaf

- What is Naïve Bayes | IBM. (n.d.). Www.ibm.com. Retrieved January 29, 2023, from https://www.ibm.com/topics/naive-bayes.

- MLNerds. (2021, July 30). Naive Bayes Classifier : Advantages and Disadvantages. Machine Learning Interviews. https://machinelearninginterview.com/topics/machine-learning/naive-bayes-classifier-advantages-and-disadvantages/

- machine learning - With the Naive Bayes classifier, why do we have to normalize the probabilities after calculating the probabilities of each hypothesis? (n.d.). Cross Validated. Retrieved January 30, 2023, from https://stats.stackexchange.com/questions/249762/with-the-naive-bayes-classifier-why-do-we-have-to-normalize-the-probabilities-a

- Tokuç, A.A. (2021). How to Improve Naive Bayes Classification Performance? | Baeldung on Computer Science. [online] www.baeldung.com. Available at: https://www.baeldung.com/cs/naive-bayes-classification-performance

- Email Spam Filtering Using Naive Bayes Classifier. (2021, June 14). Springboard Blog. https://www.springboard.com/blog/data-science/bayes-spam-filter/

- Brownlee, J. (2013, December 26). How to Evaluate Machine Learning Algorithms. Machine Learning Mastery. https://machinelearningmastery.com/how-to-evaluate-machine-learning-algorithms/

- Brownlee, J. (2020, August 2). Repeated k-Fold Cross-Validation for Model Evaluation in Python. Machine Learning Mastery. https://machinelearningmastery.com/repeated-k-fold-cross-validation-with-python/

- python - How to convert OpenDocument spreadsheets to a pandas DataFrame? (n.d.). Stack Overflow. Retrieved January 31, 2023, from https://stackoverflow.com/questions/17834995/how-to-convert-opendocument-

- classification - How to handle a zero factor in Naive Bayes Classifier calculation? (n.d.). Data Science Stack Exchange. Retrieved January 31, 2023, from https://datascience.stackexchange.com/questions/15526/how-to-handle-a-zero-factor-in-naive-bayes-classifier-calculation

- pandas documentation — pandas 1.0.1 documentation. (n.d.). Pandas.pydata.org. https://pandas.pydata.org/docs/

- Suresh, A. (2020, November 20). *What is a confusion matrix?* Medium. https://medium.com/analytics-vidhya/what-is-a-confusion-matrix-d1c0f8feda5

- *Understanding Data Attribute Types | Qualitative and Quantitative*. (2018, April 6). GeeksforGeeks. https://www.geeksforgeeks.org/understanding-data-attribute-types-qualitative-and-quantitative/

- Jayaswal, V. (2020, November 22). *Laplace smoothing in Naïve Bayes algorithm*. Medium. https://towardsdatascience.com/laplace-smoothing-in-na%C3%AFve-bayes-algorithm-9c237a8bdece

- *A Simple Guide to Data Preprocessing in Machine Learning*. (n.d.). Www.v7labs.com. https://www.v7labs.com/blog/data-preprocessing-guide

Appendices

Not included in the mark scheme and unable to fit in word count, but pertinent to the assignment.

Reason for choosing the Naives Bayes

Naive Bayes typically works well on small datasets in the training phase, making this algorithm a good option for the 30 data entries in our dataset(Tokuç, 2021). Likewise, the algorithm is typically fast and gives good results if the conditional independence for attributes is true, which we can make the assumption with our attributes being positions and how they should not influence each other with their respective attribute values.
In contrast, it does have limitations in terms of the assumption that most attributes are dependent and the zero-probability problem, where the test data for that particular class is not represented in the training data. (MLNerds, 2021),
To overcome the issue of zero probabilities we utilised Laplace smoothing by initialising all attribute value counts to have a predefined alpha value, currently hardcoded as 1 in the code(Jayaswal, V, 2020). This overcomes the zero-frequency problem. This can be seen in Fig.11.

Fig.11 Laplace smoothing in code

```
def LaplaceSmoothing(dataframe):
    #Purpose is to populate the data with an alpha value --> avoid zero frequency problem/clean data
    #Jayaswal, V. (2020, November 22). Laplace smoothing in Naïve Bayes algorithm. Medium. https://towardsdata
    alpha = 1 #value to prepopulate the data with
    PrepopulatedDataObj =  DataFrequencyStore() #will store depenedent features with alpha value
    PrepopulatedDataObj.ClassFrequency = alpha #class number(1) * alpha
    for index, row in dataframe.iterrows():
        #iterate through each row value
        for value in row[1:len(row)]:
            if not value in PrepopulatedDataObj.Attribute_Values_Dict: # Not in the dictionary?
                PrepopulatedDataObj.Attribute_Values_Dict[value] = alpha #add data with alpha value
    return PrepopulatedDataObj #return populated data object
```

Naives Bayes Uses

Typically, this algorithm is used in binary classification, e.g., in text classification determining if an email is spam or valid, depending on the words used to compromise that email(Email Spam Filtering Using Naive Bayes Classifier, 2021). In addition, an example of multi-classification Naives bayes could be adding the ‘draw’ class to our dataset and have the algorithm handle the classes ‘Win’, ‘Loss’ or ‘Draw’. This could be a future enhancement for anyone picking up this project.