**Customer Churn Analysis**

Weekly Report

Ahmedabad University

4rier Series

Prof. Mehul Raval

CSE523 - Machine Learning

**Naive Bayes from Scratch**

| class NaiveBayes:  def fit(self, X, y):  nSamples, nFeatures = X.shape  self.\_classes = np.unique(y)  nClasses = len(self.\_classes)    # calculate mean, var, and prior for each class  self.\_mean = np.zeros((nClasses, nFeatures), dtype=np.float64)  self.\_var = np.zeros((nClasses, nFeatures), dtype=np.float64)  self.\_priors = np.zeros(nClasses, dtype=np.float64)      for index, i in enumerate(self.\_classes):  Xi = X[y == i]  self.\_mean[index, :] = Xi.mean(axis=0)  self.\_var[index, :] = Xi.var(axis=0)  self.\_priors[index] = Xi.shape[0] / float(nSamples)    #predicting values from X dataset and storing it in y\_pred  def predict(self, X):  y\_pred = [self.\_predict(x) for x in X]  return np.array(y\_pred)     def \_predict(self, x):  posteriorList = []    # calculate posterior probability for each class using the help of prior knowledge  for index, i in enumerate(self.\_classes):  prior = np.log(self.\_priors[index])  posterior = np.sum(np.log(self.\_pdf(index, x)))  posterior = posterior + prior  posteriorList.append(posterior)    # return class with the highest posterior  return self.\_classes[np.argmax(posteriorList)]    #calculating the PDF using mean and variance  def \_pdf(self, class\_idx, x):  mean = self.\_mean[class\_idx]  var = self.\_var[class\_idx]  numerator = np.exp(-((x - mean) \*\* 2) / (2 \* var))  denominator = np.sqrt(2 \* np.pi \* var)  answer = numerator / denominator  return answer     # Testing # Imports  from sklearn.model\_selection import train\_test\_split  from sklearn import datasets  #accuracy calculations  def accuracy(y\_true, y\_pred):  accuracy = np.sum(y\_true == y\_pred) / len(y\_true)  return accuracy    X, y = datasets.make\_classification(  nSamples=1000, nFeatures=10, nClasses=2, random\_state=123  )  #80-20 ratio for train-test dataset.  X\_train, X\_test, y\_train, y\_test = train\_test\_split(  X, y, test\_size=0.2, random\_state=123  )  nb = NaiveBayes()  nb.fit(X\_train, y\_train)  predictedValue = nb.predict(X\_test)  print("The Naive Bayes classification's Accuracy: ", accuracy(y\_test, predictedValue)) |
| --- |

Naive Bayes classification accuracy 0.965

**References**

*sklearn.linear\_model.LogisticRegression*. (n.d.). Scikit-learn. <https://scikit-learn/stable/modules/generated/sklearn.linear_model.LogisticRegression.html>

*sklearn.naive\_bayes.GaussianNB*. (n.d.). Scikit-learn. <https://scikit-learn/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html>

*sklearn.svm.SVC*. (n.d.). Scikit-learn. <https://scikit-learn/stable/modules/generated/sklearn.svm.SVC.html>

Brownlee, J. (2021, March 16). *SMOTE for Imbalanced Classification with Python*. MachineLearningMastery.com. <https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/>