CSCI 6364 Machine Learning – Assignment 2  
Abde Manaaf Ghadiali – G29583342

*Disclaimer – Much of the Information for Question 1 and 2 has been excluded from the report due to the report’s length. Please refer to the Jupyter Notebook which is present along with this report in the zip file.*

***Question 1: Binary Classification with Custom Naive Bayes***

***Objective***

This assignment aims to construct a binary classification model employing the **Naive Bayes algorithm**. The goal is to utilize the **Myocardial Infarction Complications** dataset to forecast **Chronic Heart Failure** occurrences in patients, leveraging diverse features.

Data Source: <https://archive.ics.uci.edu/dataset/579/myocardial+infarction+complications>

***Data Loading and Preprocessing***

1. The data is fetched using the provided code snippet from the website after installing the **ucimlrepo** library via pip. Utilizing this library, we extract the dataset (ID = 579), obtaining our features, target, and metadata for column descriptions.
2. We exclude the 'ZSN\_A' feature from the independent feature list, as it essentially duplicates the target 'ZSN' feature with different binning. This leaves us with **110 features and 1700 records**.
3. Now looking at the data, we realize it has a lot of **null values**. To handle these null values, we approach it in 2 steps. Firstly, we see all the features with null values and then decide on a **threshold** (here its 0.1 i.e., any feature with more than 10% null values) and drop those features from our input data whose null values percentage is greater than the threshold. Below are the features that are dropped:

A graph with numbers and letters

Description automatically generated

1. After dropping the above features, we are left with
2. For the remaining features with null values, we apply 5 methods of imputation and save each dataset into a dictionary:
   1. **Simple Imputer with Mean Strategy**
   2. **Simple Imputer with Median Strategy**
   3. **Simple Imputer with Most Frequent Strategy**
   4. **KNN Imputer with 25 neighbors**
   5. **Iterative Imputer with Bayesian Ridge Estimator**
3. Following this, we standardize each of our datasets using three scalers and one transformer:
   1. **Standard Scaler with Power Transformer**
   2. **Min Max Scaler with Power Transformer**
   3. **Robust Scaler with Power Transformer**
4. Lastly, we split the datasets into train and test (with test ratio = 0.2, since our number of records is very small) and stratify on the target feature to get an approx. same distribution for train and test.

***Data Visualization***

1. Looking at the data, we realize that we have three types of features: **Continuous (4), Categorical (15) and Binary (72)** with a dominance of Binary features.
2. Following this, we plot **Histograms** for all the feature set side by side, to see how our imputation changed the raw dataset.

Continuous Feature (Histogram)

A graph of different sizes and colors

Description automatically generated with medium confidence

Categorical Feature (Histogram)

A graph of different sizes and shapes

Description automatically generated with medium confidence

Binary Feature (Histogram)

A graph of a number of blue bars

Description automatically generated with medium confidence

1. We also plot the **Correlation Heat Map** to see if there are any correlated features in the data. Here we see most of the features have almost zero correlation with each other.

A chart of different colors

Description automatically generated A chart of a heatmap

Description automatically generated

1. Lastly, we see the distribution of our target feature to check if the data is balanced or imbalanced. Upon inspection we realize the distribution of class is: **0 – 1306 Records (77%)** and **1 – 394 Records (23%)**

***Model Development***

1. We start the model development process by first selecting the important features. We employ 3 different **feature selection** methods and then apply **PCA (dimensionality reduction)** to reduce the feature space even more. We apply feature selection because keeping all the features some of which might not be proving useful in the final model prediction and just increasing the error.
2. Our feature selection process first removes the **highly correlated features** (at a **threshold of 0.8**) (using **Pearson’s Correlation Coefficient**).
3. Next, we use **Point Bi-Serial Correlation** to calculate Coefficients between features and target to see if any of the features are correlated to the target. We also get the **P-Value Statistic** based on which we filter out the features and keep only those features whose P-Value is less than **0.05**.

A screenshot of a computer

Description automatically generated

1. After dropping Correlated Features, we apply two methods of feature selection: **Recursive Feature Elimination (RFE)** and **Sequential Feature Selection (Forward) (SFS)**. The estimator for both is **Random Forest Classifier** and we selected the best 15 Features.
2. On those 15 features we apply **PCA (Dimensionality Reduction)** to further reduce our feature space to only the most important uncorrelated principal components. This gives us our final dataset with **10 features** in each data subset for model building.
3. The last step before model building is **balancing the data**. Since we know our target feature is **imbalanced**, we need to employ some sampling techniques to balance out the data by creating synthetic data or removing records from our data. We tried two balancing techniques: **SVMSMOTE (for Over Sampling)** and **Repeated Edited Nearest Neighbors (RENN) (for Under Sampling)**.
4. Finally for the model building process we have **15 data subsets**. We apply **Gaussian Naïve Bayes Classifier** model from **Scikit-Learn** Library on all our data subsets.

***Performance Analysis***

Below is the Final **Model Evaluation Metric** Table along with the **ROC Curve**, **Training Vs Test Accuracy Curve**, and the **Confusion Matrix**.

Table for RFE Selected Features on Gaussian Naïve Bayes Classifier:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Data Subset | *Train Accuracy* | *Test Accuracy* | *ROC* | *Precision* | *Recall* | *F-1 Score* |
| **Standard Simple Mean** | 0.681 | 0.665 | 0.619 | 0.303 | 0.342 | 0.321 |
| **Standard Simple Median** | 0.686 | 0.674 | 0.603 | 0.278 | 0.253 | 0.265 |
| **Standard Simple Most Frequent** | 0.686 | 0.674 | 0.603 | 0.278 | 0.253 | 0.265 |
| **Standard KNN** | 0.674 | 0.700 | 0.590 | 0.283 | 0.190 | 0.227 |
| **Standard Iterative** | 0.674 | 0.703 | 0.604 | 0.323 | 0.253 | 0.284 |
| **Min Max Simple Mean** | 0.681 | 0.665 | 0.619 | 0.303 | 0.342 | 0.321 |
| **Min Max Simple Median** | 0.686 | 0.674 | 0.603 | 0.278 | 0.253 | 0.265 |
| **Min Max Simple Most Frequent** | 0.686 | 0.674 | 0.603 | 0.278 | 0.253 | 0.265 |
| **Min Max KNN** | 0.674 | 0.700 | 0.590 | 0.283 | 0.190 | 0.227 |
| **Min Max Iterative** | 0.674 | 0.703 | 0.604 | 0.323 | 0.253 | 0.284 |
| **Robust Simple Mean** | 0.681 | 0.665 | 0.619 | 0.303 | 0.342 | 0.321 |
| **Robust Simple Median** | 0.686 | 0.674 | 0.603 | 0.278 | 0.253 | 0.265 |
| **Robust Simple Most Frequent** | 0.686 | 0.674 | 0.603 | 0.278 | 0.253 | 0.265 |
| **Robust KNN** | 0.674 | 0.700 | 0.590 | 0.283 | 0.190 | 0.227 |
| **Robust Iterative** | 0.674 | 0.703 | 0.604 | 0.323 | 0.253 | 0.284 |

ROC Curve, Confusion Matrix, and Model Learning Curve

A chart of a color spectrum

Description automatically generated with medium confidence

***Reflection and Insights***

1. **Consistency in Performance**: Across various preprocessing methods and scaling techniques (Standard, Min Max, Robust), there is a remarkable consistency in both train and test accuracies, ROC, precision, recall, and F-1 scores. This consistency suggests that the choice of preprocessing method may have a limited impact on the model's performance.
2. **Impact of Imputation Strategy**: Notably, the choice of imputation strategy (Mean, Median, Most Frequent) does not significantly affect the model's performance, as evidenced by the similarity in performance metrics across different imputation methods.
3. **Generalization Capability**: The model's ability to achieve comparable performance on both training and test datasets suggests that it can generalize well to unseen data. This is crucial for real-world applications where the model needs to make accurate predictions on new, previously unseen instances.
4. **Handling Imbalanced Data**: Despite the imbalance in the target feature, the model demonstrates the capability to predict chronic heart failure occurrences effectively. While there may be some room for improvement in metrics related to the minority class, the model's ability to maintain consistent performance across different subsets indicates its efficacy in handling imbalanced data.
5. **Room for Improvement**: While the model demonstrates promising predictive power, there is always room for improvement. Further exploration of feature engineering techniques, model selection, and hyperparameter tuning could potentially enhance the model's performance and predictive power. Additionally, incorporating domain knowledge or exploring alternative machine learning algorithms may lead to better predictive outcomes.

***Question 2: Emotion Analysis using SVM and K-Means***

***Objective***

Develop an understanding of emotion recognition in text using **Support Vector Machines (SVM)** for classification and **K-Means clustering** for pattern discovery The goal is to utilize the **Emotion Lines** dataset to predict the different emotions based on the sentence utterance by a speaker (in text format).

Data Source: <https://blackboard.gwu.edu/bbcswebdav/pid-13804467-dt-content-rid-126822844_2/xid-126822844_2>

***Data Preparation***

1. The Emotion Lines Data in its Raw format was JSON list of lists of dictionaries. We have 3 such JSON files for training, validation (dev) and test. Each dictionary had four key-value pairs with the keys being, speaker, utterance, emotion, annotation. We convert each list of dictionaries to a Pandas Dataframe and concatenate these data frames to create one Dataframe for each train, dev, test.

***A screenshot of a black screen

Description automatically generated***

1. We see that our Raw data consists of 10516 Training Records, 1178 Validation Records and 2764 Testing Records. Also, there are a few Hexadecimal Values (mainly ‘) which have not been decoded properly. We replace those hexadecimal values with the appropriate ASCII value.

A screenshot of a computer screen

Description automatically generated

1. Next, we convert all the strings to Lower Case and fix any Contractions present in the text. Following this we tokenize each sentence into Word Token which breaks the sentences into words.

A screenshot of a computer

Description automatically generated

1. From here, we divide the dataset into two streams, one where we remove the stop words and the other where we do not remove the stop words. On both the data features, we apply Lemmatization technique (with Parts of Speech Tagging) to transform the words into its root form.

A screen shot of a computer

Description automatically generated

1. Lastly, we employ TF-IDF Vectorization to get the statistic of a word in the text relative to all the text. We apply TF-IDF Vectorization fit method to the training data and transform the validation and test data. Along with TF-IDF on the Text data, we apply Label Encoder on the target data to get numeric encoding of the emotions.

***A black screen with white text

Description automatically generated***

***SVM for Emotion Classification***

1. After we have completed the TF-IDF Vectorization, we start building the standard default SVM Model. Initially we built two models. One for token with stop words removed and for text without stop words removed. This will be our baseline model.

|  |  |  |
| --- | --- | --- |
| Data Subset | Cross Validation Train Accuracy | Cross Validation Test Accuracy |
| Data With Stop Words | 0.9627 | 0.5587 |
| Data Without Stop Words | 0.9404 | 0.5398 |

1. After getting the Baseline, we employ Grid Search to find the best parameters for SVC and build out two models based on the best parameters to get out final model. Since our target is imbalanced, we add class weights as a parameter to balance the weights of each class.
2. After Executing Grid Search, we realize that the Base Line is the best SVM Model as the Grid Search also provided with the same parameters for SVM.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Subset | Train Accuracy | Validation Accuracy | Test Accuracy |
| Data With Stop Words | 0.960 | 0.526 | 0.567 |
| Data Without Stop Words | 0.937 | 0.510 | 0.560 |

Confusion Matrix and Accuracy Plot for Data with Stop Words

A graph and chart with different colored lines

Description automatically generated with medium confidence

Confusion Matrix and Accuracy Plot for Data without Stop Words

A graph of a graph and a graph of a graph

Description automatically generated with medium confidence

***K-Means Clustering***

1. We assume twice the number target classes as the maximum number of clusters and iterate over the range to create the K-Means Model. We then save the inertia\_ attribute of the model in a list. This becomes our distortions as inertia\_ is nothing but sum of squared distances of samples to their closest cluster center.
2. We apply this approach on both our data set and plot the graph of these distortions to get the optimal number of clusters using the elbow method.

A graph with a line

Description automatically generated A graph with a line

Description automatically generated

1. Once we have the optimal number of clusters, we apply Truncated SVD on the input data sets to reduce the feature space of the data. We get the 2 principal components and run a K-Means Model on this data with the optimal number of clusters.
2. We then plot the 2 principal components and the cluster mapping with the centroid of each cluster.

A diagram of clustering

Description automatically generated A diagram of a diagram

Description automatically generated with medium confidence

***Model Insights***

1. From the SVM Model Stats and the K-Means distortions and clusters, we can infer that Unsupervised Learning is visually performing better than SVM model. Our K-Means can detect the 7 different clusters for the different emotions.
2. Secondly Our SVM Model is Overfitting the Data as the Training Accuracy is wat more than the Validation or Testing Accuracy.
3. Lastly, we can say that SVM model might not be the best fit model for this task as it cannot detect the non-linearity of the data which is in contrast to the K-Means Clustering method which is able to capture the different clusters.