UNIVERSITY OF BRIGHTON

REPORT TITLE: - BREAST CANCER PREDICTION

MODULE TUTOR: - Saeed Malekshahi

MODULE CODE: - CI513 Intelligence system 2

STUDENT ID: - 21839630

SUBMISSION DATE: - 22/05/2023

Table of Contents:

1. Introduction

1.1 Objective

1.2 Dataset Description

1.3 Supervised Learning and Breast Cancer Diagnosis

2. Implementation of AI Pipeline

2.1 Data Collection

2.2 Data Exploration

2.3 Data Preparation and Processing

2.4 Model Selection

2.5 Model Training and Testing

2.6 Conclusion

3. Data Importing

3.1 Code for Importing Data

3.2 Figure 1: Importing Data

4. Data Analysis and Cleaning

4.1 Null Values

4.2 Checking the Distribution of Features

4.3 Figure 2: Distribution Range of Features 1

4.4 Figure 3: Distribution Range of Features 2

4.5 Figure 4: Distribution Range of Features 3

4.6 Converting Diagnosis into Benign and Malignant

4.7 Figure 5: Benign vs Malignant

4.8 Checking the Correlation between Features

4.9 Figure 6: Correlation between Features

5. Data Preparation

5.1 Splitting the Data

5.2 Train-Test Split

5.3 Standardizing the Data

5.4 Figure 7: Data Preparation

6. Model Implementation

6.1 Logistic Regression Model

6.1.1 Model Training

6.1.2 Prediction

6.1.3 Confusion Matrix

6.1.4 Classification Report and Accuracy

6.1.5 Figure 8: Logistic Regression

6.2 XGBoost Classifier

6.2.1 Model Initialization

6.2.2 Model Training and Hyperparameter Tuning

6.2.3 Prediction

6.2.4 Confusion Matrix

6.2.5 Classification Report

6.2.6 Figure 9: XGBoost Classifier

6.2.7 Figure 10: XGBoost Classifier 2

6.3 K-Nearest Neighbors (KNN) Classifier

6.3.1 Model Initialization

6.3.2 Defining the Parameter Grid for Hyperparameter Tuning

6.3.3 Performing Grid Search with Cross-Validation

6.3.4 Obtaining the Best Parameters and Best Score

6.3.5 Making Predictions on the Test Set

6.3.6 Evaluating the Model Performance

6.3.7 Figure 11: KNN Classifier

6.3.8 Figure 12: KNN Classification Report

6.4Decision Tree

6.4.1. Importing the required modules

6.4.2 Instantiating and training the Decision Tree classifier

6.4.3. Making predictions

6.4.4. Confusion matrix and classification report

6.5Naive Bayes

6.5.1. Importing the required libraries

6.5.2. Model training

6.5.3. Prediction

6.5.4. Confusion matrix visualization

6.5.5. Model evaluation

7.Selection of Best Model

7.1. Accuracy

7.2. Sensitivity and Specificity

7.3. Precision

7.4. Comparison of FN and FP between models

7.5. Conclusion on the best model

8. Top 10 Features

9.conclusion

1.Introduction

1.1 Objective

The primary goal of my AI/Pipeline is to provide precise and reliable predictions on the likelihood of an individual being diagnosed with breast cancer. The pipeline operates by processing the relevant input data and generating the corresponding probability of breast cancer. The chosen dataset for this project is the Breast Cancer Dataset, which includes comprehensive information on the characteristics of cell nuclei present in the breast. The dataset contains various attributes suitable for accurate classification and prediction purposes.

1.2 Dataset Description

The Breast Cancer dataset is a supervised learning dataset that has labeled data classes for each attribute. This means that an AI pipeline can be implemented on it with the help of supervised learning. In this dataset, the labeled data can be differentiated into two categories: malignant and benign tumors.

1.3 Supervised Learning and Breast Cancer Diagnosis

The process of supervised learning involves training and testing the model using labeled data. This helps the algorithm to understand and recognize the patterns in the dataset, and predict the diagnosis based on different features of the tumor, such as radius, texture, and compactness. By analyzing these features, the model can provide a diagnosis of whether a tumor is cancerous or non-cancerous.

2. Implementation of AI Pipeline

The prosecution of the AI and Pipeline involves several ways, which are described below: -

* 1. Data Collection: - The first step is to collect the needed source of breast cancer dataset that will be used for the model.

2.2 Data Exploration: - This process involves exploring the dataset to determine which features can be used for the model and which cannot. (Dong et al., 2014)

2.3 Data Preparation and Processing: - The Breast cancer dataset may contain raw data, missing values, and unwanted data, which must be removed and prepared for the coming stage of the channel.

* 1. Model Selection: - The process of opting the most suitable model for the given dataset is carried out.
  2. Model Training and Testing The model is trained and tested using the dataset to ensure that it provides accurate prognostications.
  3. Conclusion: - After testing all the model. checking which model gives the best accurate.

output for the data provide.

3. Data Importing

The code reads the data from an external source, specifically from a CSV file named "Cancer\_Data.csv". It uses the pandas library's read\_csv() function to load the data into a DataFrame called "data".

A screenshot of a computer

Description automatically generated with low confidence

Figure 1.importing data

4. Data Analysis and Cleaning

4.1 Null Values

The code checks the dataset for any issues or exceptions that may affect processing.

Checking for null values in the code removes two columns, "id" and "Unnamed: 32", from the Data Frame because they contain null values and are not relevant for the prediction.

4.2 Checking the Distribution of Features

Checking the distribution range of attributes in The code creates a 3x3 grid of subplots using matplotlib's subplots() function. It then reshapes the subplot array into a 1-dimensional array using the ravel() method. Next, it defines the range of columns (attributes) to be analysed, specifically columns 3 to 11, and assigns them to the variable "cols". The code iterates over the indices and corresponding axes in the 1-dimensional array, plots a histogram of the corresponding column from the Data Frame "data" using the hist() method of each Axes object, and adjusts the layout of the subplots using tight\_layout() to ensure proper spacing.

A screenshot of a graph

Description automatically generated with medium confidence

Figure 2 Distribution range of features

A screenshot of a computer screen

Description automatically generated with low confidence

Figure 3Distribution range of features

A screenshot of a computer screen

Description automatically generated with low confidence

Figure 4. distribution range of features 3

4.6 Converting Diagnosis into Benign and Malignant

To visualize the distribution of the target variable, which represents the diagnosis of breast cancer. Mapping the Diagnosis Values: The code maps the values in the "diagnosis" column of the dataset. It replaces the categorical values 'B' (benign) with 0 and 'M' (malignant) with This mapping converts the categorical variable into a numerical representation, which is often beneficial for further analysis and modelling.

After mapping I have Created a bar plot After mapping the diagnosis values, the code uses the plt.bar() function from the matplotlib library to create a bar plot. The plot visualizes the count of each category in the "diagnosis" column. The x-axis represents the categories "Benign" and "Malignant", while the y-axis represents the count of occurrences for each category.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 5. Benign VS Malignant

Benign: The count for the benign category represents the number of instances in the dataset classified as non-malignant, which is labelled as 0. In this case, the count is 350.

Malignant: The count for the malignant category represents the number of instances in the dataset classified as cancerous, which is labelled as 1. The count for this category is 200.

The observation of an imbalanced label distribution indicates that the dataset contains a higher number of instances labelled as benign compared to malignant.

4.8 Checking the Correlation between Features

In order to explore the relationship between the "diagnosis" attribute and other attributes in the dataset, a correlation analysis was conducted. The corr() function was applied to calculate the correlation coefficients between the "diagnosis" attribute and all other attributes in the dataset. This resulted in a correlation matrix that quantifies the linear relationship between pairs of attributes. The correlation values related to the "diagnosis" attribute were then selected and sorted in descending order using the sort\_values() function. This sorting process arranged the attributes based on their correlation strength with the "diagnosis" attribute, placing the attributes with the highest correlation at the top.

A screen shot of a graph

Description automatically generated with medium confidence

Figure 6 Correlation between the features

To visually represent the correlations, a bar plot was employed. The plot() function was utilized with the argument kind='bar', which generated a bar chart. Each attribute was plotted on the x-axis, while the corresponding correlation value was displayed on the y-axis. Longer bars indicated stronger positive or negative correlations.

Upon analysing the correlation plot, it was determined that none of the features exhibited a high correlation with the "diagnosis" attribute. As a result, no features were considered for removal based on their correlation strength.

5. Data Preparation

5.1 Splitting the Data

The dataset is divided into features and labels. The features are obtained by dropping the "diagnosis" column from the original dataset using the drop() function with axis=1. The remaining columns are assigned to the variable X as the feature values. The labels, representing the "diagnosis" column, are assigned to the variable y.

5.2 Train-Test Split

The dataset is split into training and testing sets using the train\_test\_split() function from the sklearn.model\_selection module. The feature and label data (X and y) are passed as input along with the test\_size parameter set to 0.2, indicating that 20% of the data will be allocated for testing. The stratify parameter is also set to y, ensuring that the class distribution is maintained in both the training and testing sets.

Standardizing the Data:-

The feature data is standardized to have zero mean and unit variance. This is achieved using the StandardScaler() class from the sklearn.preprocessing module. First, the fit\_transform() method is applied to the training set X\_train, which computes the mean and standard deviation of the training data and scales it accordingly. Then, the transform() method is used on the testing set X\_test to apply the same scaling parameters obtained from the training set.

By splitting the data into features (X) and labels (y), and performing a train-test split, we can indeed infer that this is a supervised learning problem.

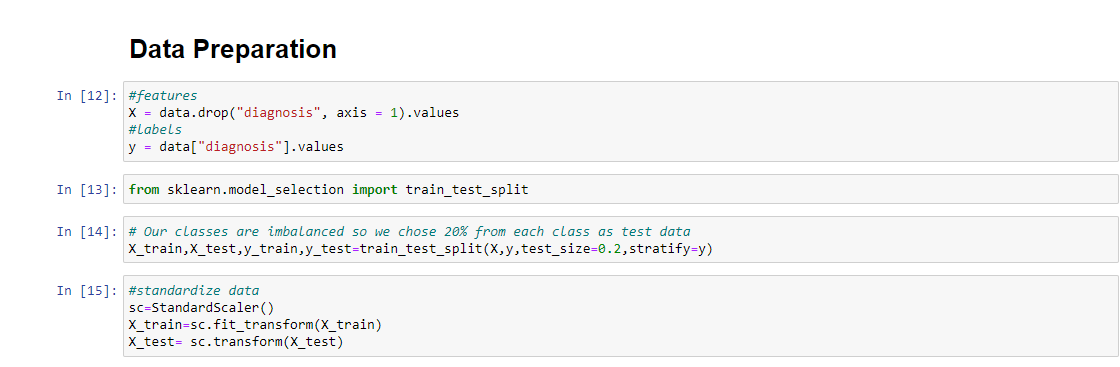


Figure 7. DATA PREPARATION

6. Model Implementation

For implementing an model we need to select models which suits with the prediction for the dataset and it is should be for supervised learning. I have selected five models and trained and fitted them for prediction and the five models are: -

1. Logistic regression
2. XGBoost classifier
3. K-nearest-neighbors
4. Decision tree
5. Random forest

6.1 Logistic Regression Model

The logistic regression model is a statistical model used for binary classification tasks. (Zou et al., 2019) It predicts the probability of a binary outcome, such as whether a breast cancer diagnosis is benign or malignant. One of the advantages of logistic regression is its interpretability, as it can indicate the impact of each feature on the predicted probability. (R. Rekha and K.L. Vinoci, 2023)

6.1.1 Model Training

In this implementation, a logistic regression model (lr) was instantiated with random\_state=0 for reproducibility. The model was then trained by fitting it to the training data (X\_train) and corresponding labels (y\_train).

6.1.2 Prediction

After training the logistic regression model, it was used to make predictions on the test set (X\_test). The prediction method was applied to the trained model to obtain predicted labels for each instance in the test set.

6.1.3 Confusion Matrix

To assess the performance of the logistic regression model, a confusion matrix was computed by comparing the true labels (y\_test) with the predicted labels (predicted1). The confusion matrix provides insights into the number of true positives, true negatives, false positives, and false negatives. It was visualized using a heat map plot to aid in understanding the model's performance.

6.1.4 Classification Report and Accuracy

The classification\_report function was utilized to generate a comprehensive report of the model's performance. The report includes metrics such as precision, recall, F1-score, and support for each class (benign and malignant). Additionally, the accuracy of the model was calculated by summing the true positives and true negatives and dividing by the total number of instances.

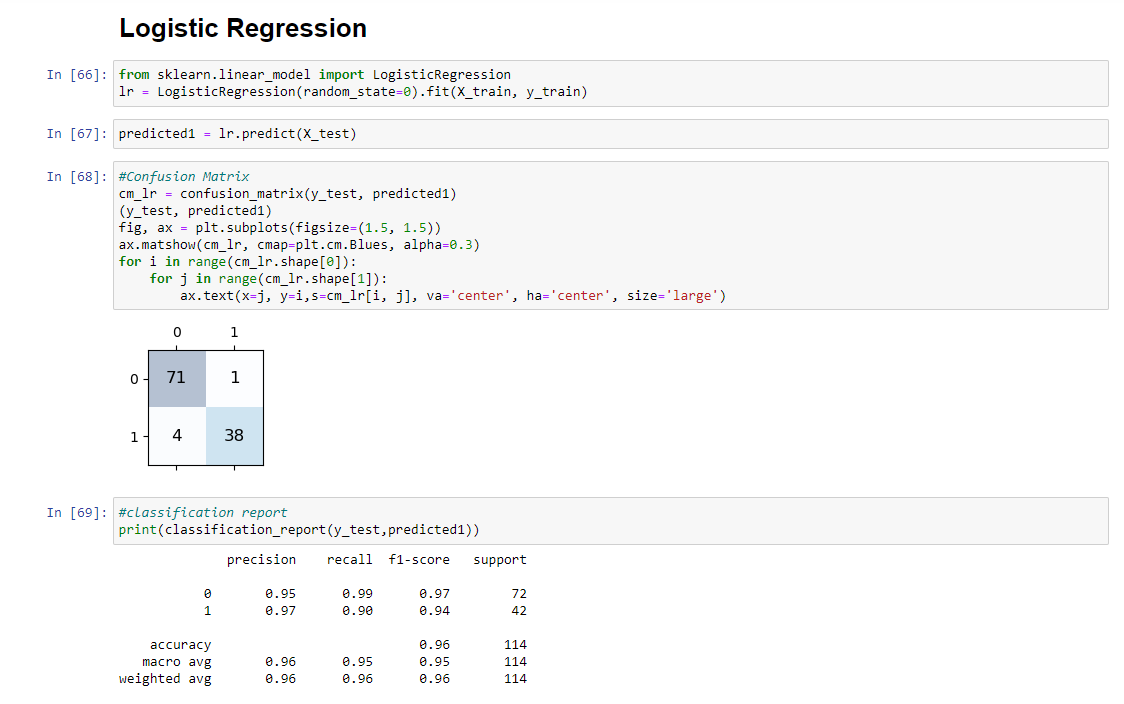


Figure 8. Logistic regression

6.2 XGBoost Classifier

It is a powerful machine learning algorithm known for its effectiveness in handling structured data and delivering high predictive performance (Irawati and Zakaria, 2021). Its from family of gradient boosting algorithms and is particularly well-suited for classification. It has high accuracy, feature importance, handling imbalanced data and some regularization which is best used for my breast cancer data set.

6.2.1 Model Initialization

The XGBoost classifier model is imported from the xgboost.sklearn module. The hyperparameters for the XGBoost classifier are defined in the parameters dictionary. These hyperparameters specify various aspects of the model, such as learning rate, maximum depth of trees, subsampling ratio, and regularization parameters.

6.2.2 Model Training and Hyperparameter Tuning

A new instance of the XGBoost classifier model is created (model = XGBClassifier()). Next, a randomized search is performed to find the optimal combination of hyperparameters for the model. The RandomizedSearchCV function is used with the XGBoost classifier model, the defined parameters, 2-fold cross-validation (cv = 2), and -1 for n\_jobs to utilize all available CPU cores. The training data (X\_train and y\_train) is used for the hyperparameter tuning.

6.2.3 Prediction

After the hyperparameter tuning is completed, the trained XGBoost classifier model (xgb\_grid) is used to make predictions on the test set (X\_test). The predict method is applied to obtain the predicted labels (predicted2).

6.2.4 Confusion Matrix

A confusion matrix is computed by comparing the true labels (y\_test) with the predicted labels (predicted2). The confusion\_matrix function from sklearn.metrics is used for this purpose. The resulting confusion matrix is then visualized as a heat map plot using matplotlib.

A screenshot of a computer

Description automatically generated

Figure 9. XGB CLASSIFIER

6.2.5 Classification Report

The classification\_report function from sklearn.metrics is used to generate a comprehensive report of the XGBoost classifier model's performance. The report includes metrics such as precision, recall, F1-score, and support for each class (benign and malignant). It provides insights into the model's predictive accuracy for different classes.

A screenshot of a computer program

Description automatically generated with medium confidence

Figure 10. XGB CALSSIFIER 2

.

6.3 K-Nearest Neighbors (KNN) Classifier

The K-Nearest Neighbors (KNN) classifier is a simple yet effective supervised learning algorithm used for both classification and regression tasks. The KNN algorithm works by storing the entire training dataset in memory and, during the prediction phase, calculates the distance between a new test instance and all the training instances(C, 2021). It then selects the K nearest neighbors (i.e., the K training instances with the smallest distances) and assigns the majority class among those neighbors as the predicted class for the new instance.

By tuning the hyperparameters of KNN, such as the value of K, different distances metrics, and weighting schemes, you can optimize the model's performance on the breast cancer dataset.

* + 1. Model Initialization

Importing the necessary libraries: The code begins by importing the required libraries, including KNeighborsClassifier from sklearn.neighbors, which is the implementation of the KNN algorithm in scikit-learn.Creating the KNN classifier is an instance of the KNN classifier is created with the default parameters by calling KNeighborsClassifier().

6.3.2 Defining the Parameter Grid for Hyperparameter Tuning

A list k\_list is created, which contains the values of the n\_neighbors hyperparameter (i.e., the number of neighbors to consider) that will be evaluated during the grid search. The parameter grid k\_values is then created as a dictionary with the key 'n\_neighbors' and the list of values as its corresponding value.

6.3.3 Performing Grid Search with Cross-Validation

The GridSearchCV object grid is instantiated with the KNN classifier and the parameter grid. It is configured to use 5-fold cross-validation (cv = 5), evaluate models based on accuracy (scoring = 'accuracy'), and utilize all available CPU cores (n\_jobs = -1). The fit method is called on grid with the training data (X\_train and y\_train).

A screenshot of a computer program

Description automatically generated with medium confidence

Figure 11. KNN

6.3.4 Obtaining the Best Parameters and Best Score

After the grid search is completed, the best parameters found during the search are accessed using grid.best\_params\_, and the corresponding best score is obtained using grid.best\_score\_. These values provide insights into the optimal hyperparameter settings for the KNN classifier on the breast cancer dataset.

6.3.5 Making Predictions on the Test Set

The trained KNN classifier with the best parameters (obtained from the grid search) is used to predict the labels for the test set (X\_test). The predictions are stored in the variable predicted3.

6.3.6 Evaluating the Model Performance

The code calculates the confusion matrix using confusion\_matrix from sklearn.metrics, and visualizes it as a heat map using matplotlib.pyplot. The classification\_report function is used to generate a comprehensive report containing various evaluation metrics (precision, recall, F1-score, etc.) for both classes (benign and malignant).

A screenshot of a computer program

Description automatically generated with medium confidence

Figure 12. KNN CLASSIFICATION REPORT

6.4Decision Tree

The Decision Tree classifier uses a tree-based model to make predictions based on a set of if-then-else decision rules derived from the training data. It splits the feature space into distinct regions based on the feature values and their importance in predicting the target variable. (Venkatasubramaniam et al., 2017) The resulting decision tree can be used for classification by traversing the tree based on the feature values of a new instance until reaching a leaf node that corresponds to a predicted class label.

The code trains the Decision Tree classifier on the breast cancer dataset and evaluates its performance using the confusion matrix and classification report. The confusion matrix provides insights into the model's ability to correctly classify instances, while the classification report presents a detailed summary of the model's precision, recall, F1-score, and support for each class.

6.4.1. Importing the required modules

• from sklearn.tree import DecisionTreeClassifier: This imports the DecisionTreeClassifier class from the scikit-learn library, which provides the implementation of the decision tree algorithm.

• from sklearn.metrics import confusion\_matrix: This imports the confusion\_matrix function from scikit-learn, which is used to calculate the confusion matrix.

• import matplotlib.pyplot as plt: This imports the matplotlib library for data visualization.

6.4.2 Instantiating and training the Decision Tree classifier.

• dt = DecisionTreeClassifier(): This creates an instance of the DecisionTreeClassifier.

• dt = dt.fit(X\_train, y\_train): This fits the Decision Tree classifier to the training data (X\_train and y\_train), allowing it to learn the patterns and relationships between the features and the target variable.

3. Making predictions:

• predicted4 = dt.predict(X\_test): This uses the trained Decision Tree classifier to predict the labels for the test data (X\_test), generating the predicted labels (predicted4).

A screenshot of a computer

Description automatically generated

Figure 13. Decision tree

6.4.4. Confusion matrix and classification report

• conf\_matrix = confusion\_matrix(y\_test, predicted4): This computes the confusion matrix by comparing the true labels (y\_test) with the predicted labels (predicted4).

• Visualizing the confusion matrix:

• fig, ax = plt.subplots(figsize=(1.5, 1.5)): This creates a plot with a specified size for the confusion matrix visualization.

• Visualizing the confusion matrix using a heatmap plot: This code block creates a heatmap plot to visualize the confusion matrix. Each cell in the heatmap represents the count of true positive, true negative, false positive, and false negative predictions.

• print(classification\_report(y\_test, predicted4)): This prints a comprehensive report of the model's performance, including metrics such as precision, recall, F1-score, and support for each class (benign and malignant).

6.5Naive Bayes

The Naive Bayes algorithm is a probabilistic classifier that is commonly used for classification tasks, including the breast cancer dataset (Zhang and Gao, 2011). It can be used to predict the probability of a sample belonging to a specific class and By comparing the probabilities, the algorithm predicts the class label for a given sample.

6.5.1. Importing the required libraries

• from sklearn.naive\_bayes import GaussianNB: This imports the Gaussian Naive Bayes classifier from the scikit-learn library.

• from sklearn.metrics import confusion\_matrix: This imports the confusion\_matrix function from scikit-learn for evaluating the model's performance.

• import matplotlib.pyplot as plt: This imports the matplotlib library for visualizing the confusion matrix.

6.5.2. Model training

• nb = GaussianNB(): This creates an instance of the Gaussian Naive Bayes classifier.

• nb.fit(X\_train, y\_train): This trains the classifier on the training data X\_train and corresponding labels y\_train.

6.5.3. Prediction

• predicted5 = nb.predict(X\_test): This uses the trained classifier to make predictions on the test data X\_test and assigns the predicted labels to predicted5.

A screenshot of a computer

Description automatically generated

Figure 14. Naive bayes

6.5.4. Confusion matrix visualization

• conf\_matrix = confusion\_matrix(y\_test, predicted5): This calculates the confusion matrix by comparing the true labels y\_test with the predicted labels predicted5.

• fig, ax = plt.subplots(figsize=(1.5, 1.5)): This creates a figure and axis object for plotting the confusion matrix.

• ax.matshow(conf\_matrix, cmap=plt.cm.Blues, alpha=0.3): This visualizes the confusion matrix as a color-coded matrix.

• ax.text(...): This adds text annotations to the cells of the confusion matrix, displaying the count of each class.

• plt.show(): This displays the confusion matrix plot.

6.5.5. Model evaluation

• print(classification\_report(y\_test, predicted5)): This computes and prints a comprehensive classification report, including metrics such as precision, recall, F1-score, and support for each class (benign and malignant).

7.Selecting of Best Model

The best way of selecting the best model for our breast cancer data set is by comparing the factors which we can use as key aspect to evaluate.

* FN = the model predicts a negative outcome (no cancer) for a patient who has breast cancer
* FP = the model predicts a positive outcome (cancer) for a patient who does not have breast cancer
* TP = the model predicts a positive outcome (cancer) for a patient who actually has breast cancer.
* TN = the model predicts a negative outcome (no cancer) for a patient who does not have breast cancer.

7.1. Accuracy

Comparing the all the accuracy provided by the models and evaluating which model has the highest accuracy which gives the percentage of correct prediction and provides performance of the model.

The formula for the accuracy is

Accuracy = (Number of correct predictions) / (Total number of predictions)

* Number of correct predictions = True Positive/ True Negative
* Total number of predictions = True Positive + False Negative + True Negative + False Positive
* Accuracy = (True Positive/ True Negative)/( True Positive + False Negative + True Negative + False Positive)
* Accuracy = (TP/TN)/(TP+FP+TN+FN)

7.2. Sensitivity and Specificity

They are also known as recall or true positive rate and specificity of the models. Sensitivity measures the ability of the model to correctly identify malignant cases, while specificity measures the ability to correctly identify benign cases. A good model should have high sensitivity and specificity values.

Sensitivity (True Positive Rate or Recall) = TP / (TP + FN)

Specificity (True Negative Rate) = TN / (TN + FP)

7.3. Precision

It is a performance metric that measures the accuracy of positive predictions made by a model. It quantifies the proportion of correctly predicted positive instances (true positives) out of all instances predicted as positive (true positives plus false positives).

Precision = TP/ (TP + FP)

7.4. Comparison of FN and FP between models

Comparing false negatives (FN) and false positives (FP) between models is important because it helps us understand the type of errors each model is making and the impact of those errors. By comparing FN and FP between models, we can assess the trade-off between missing positive instances and making false positive predictions. Different applications may prioritize reducing one type of error over the other.

We consider that the model with lowest FN and FP is used as the best model. So, among the all the models the knn has got the lowest FN and FP with FN=0 AND FP=3

A picture containing text, screenshot, diagram, line

Description automatically generated

Considering all this we conclude that the KNN is the best model.

8. Top 10 Features

For selecting the top 10 features of the dataset I have defined the target variables features = X: This assigns the feature matrix X and target = y: This assigns the target variable y. which is then performs the feature selection by an instance of the SelectKBest class, specifying the scoring function as chi-squared (chi2) and the number of top features to select (k=10) and it fits the SelectKBest instance to the features and target variables, computing the scores and p-values for each feature based on the chi-squared test.

featureScores Data frame has been created for storing the feature scores computed by the chi-squared test. The scores are extracted from fit.scores\_, and the index is set to the feature names (presumably obtained from data.columns[1:]), and the column name is set as 'Chi Squared Score'. They are presented in data frame in descending order on chi squared square.

A screenshot of a computer program

Description automatically generated with medium confidence

9.Conclusion

In medical data analysis, the goal is to identify true positive results with high accuracy to make accurate diagnoses and improve patient outcomes. Each model has its strengths and weaknesses, and the choice of the most appropriate model depends on the specific characteristics of the data and the problem at hand. However, in general, models that have higher true positive rates (TPRs) are more desirable, as they correctly identify more cases of true positives. It's important to note that in medical data analysis, false negatives are particularly costly, as they can result in missed diagnoses or delayed treatment, which can have serious consequences for patients. Therefore, models that prioritize high TPRs, even at the cost of lower specificity or more false positives, may be preferred in medical applications. In our case K-Nearest neighbours gave less false-positives compared to the other models.

References

Zou, X., Hu, Y., Tian, Z. and Shen, K. (2019). Logistic Regression Model Optimization and Case Analysis. [online] IEEE Xplore.

R. Rekha and K.L. Vinoci (2023). Wisconsin Breast Cancer Detection Using L1 Logistic Regression.

Dong, Y., Guo, H., Zhi, W. and Fan, M. (2014). *Class Imbalance Oriented Logistic Regression*. [online] IEEE Xplore.

Steidl, M., Felderer, M. and Ramler, R. (2023). The pipeline for the continuous development of artificial intelligence models—Current state of research and practice. *Journal of Systems and Software*, p.111615.

Irawati, M.E. and Zakaria, H. (2021). Classification Model for Covid-19 Detection Through Recording of Cough Using XGboost Classifier Algorithm. *2021 International Symposium on Electronics and Smart Devices (ISESD)*.

C, C. (2021). Prediction of Heart Disease using Different KNN Classifier. *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*.

Venkatasubramaniam, A., Wolfson, J., Mitchell, N., Barnes, T., JaKa, M. and French, S. (2017). Decision trees in epidemiological research. *Emerging Themes in Epidemiology*, 14(1).

Zhang, W. and Gao, F. (2011). An Improvement to Naive Bayes for Text Classification. *Procedia Engineering*, 15, pp.2160–2164.

‌

‌

‌

‌

‌