Data Analytics Project

Research · July 2024

Problem Statement: Knowledge Representation and Insights Generation from Structured Datasets

Submitted By:
Abhishek Jha:
Ashish Kumar:
Ayushi Singh:
Ayushi Vardhan:

4th Semester
School of Computer Engineering
Kalinga Institute of Industrial Technology, Bhubneshwar

Table of Contents

INTRODUCTION

PROBLEM STATEMENT
TITLE: KNOWLEDGE REPRESENTATION AND INSIGHT GENERATION FROM STRUCTURED DATASETS

Understanding the Objective:

Develop an AI solution that processes datasets to extract and represent knowledge and generate valuable insights.

- Clean and prepare the dataset for analysis.
- Identify patterns, trends, and anomalies in the dataset.
- Derive meaningful conclusions from the data that can help in decision-making.
- Visualize the data and insights in a user-friendly manner

Problem Description:

The project addresses the challenge of extracting valuable insights from large volumes of data organizations generate.

- Organizations are overwhelmed with big data as they often lack the tools to derive insights from it.
- Effective data analysis can improve decision-making and overcome the problems faced by the organization as it transforms raw data into a format that reveals knowledge and insights effectively.

Task Breakdown:

Create an AI solution to process and analyze a provided structured dataset, represent its knowledge, and generate insights.

- Use a structured dataset from the UCI Machine Learning Repository.
- Data Preprocessing
- Knowledge Representation
- Pattern Identification
- Insight Generation

Feature Requirements:

- Data Preprocessing: Address missing values, duplicates, and errors. Convert data into usable formats, such as normalizing numerical data or encoding categorical variables. Use techniques like feature selection or dimensionality reduction to simplify the dataset.
- Knowledge Representation: Create charts and graphs (bar charts, pie charts, line graphs, scatter plots) to summarize and illustrate key data points.
- Pattern Identification: Identify and analyze trends over time or within categories. Find and explain outliers or unusual patterns in the data. Determine relationships between different variables in the dataset.
- Insight Generation: Translate identified patterns into insights that can inform decision-making, such as predictions, recommendations, or strategic guidance. Present insights in a clear, concise manner that is easy to understand for users, possibly through dashboards or reports.

Scalability and User Interface:

- Scalability: Design the solution to handle datasets of various sizes and complexities. Ensure the solution can scale from small datasets to large, complex ones without significant performance degradation.
- User-friendly Interface: Create an intuitive interface for users to interact with the data and insights. Enable interactive data exploration (e.g., filter, zoom, hover-over details). Ensure insights are presented in a manner accessible to users with varying levels of data literacy.

Implementation and Delivery:

Plan and execute the development of the solution.

- Choose appropriate AI and data visualization tools (e.g., Python, R, Tableau).
- Break the project into phases such as data acquisition, preprocessing, analysis, and visualization.
- Test the solution with different datasets to ensure accuracy, performance, and scalability.
- Provide documentation and training to help users understand and utilize the solution effectively.

DATASET DESCRIPTION

Breast Cancer - UCI Machine Learning Repository
Link: https://archive.ics.uci.edu/dataset/14/breast+cancer

This breast cancer domain was obtained from the University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia. This is one of three domains provided by the Oncology Institute that has repeatedly appeared in the machine learning literature.

- Objective: The dataset aims to distinguish between patients with benign and malignant breast tumors based on various attributes measured from cell nuclei.
- Context: This dataset is typically used for training machine learning models to predict the likelihood of breast cancer based on the input features.

- Sources:
  - Matjaz Zwitter & Milan Soklic (physicians)
    Institute of Oncology
    University Medical Center
    Ljubljana, Yugoslavia
  - Donors: Ming Tan and Jeff Schlimmer (Jeffrey.Schlimmer@a.gp.cs.cmu.edu)
  - Date: 11 July 1988

- Number of Instances: 286
  201 instances of one class
  85 instances of another class
- Number of Attributes: 9 + the class attribute(some of which are linear and some are nominal.)

◦ Attribute Information:

1. Class: no-recurrence-events, recurrence-events
2. age: 10-19, 20-29, 30-39, 40-49, 50-59, 60-69, 70-79, 80-89, 90-99.
3. menopause: lt40, ge40, premeno.
4. tumor-size: 0-4, 5-9, 10-14, 15-19, 20-24, 25-29, 30-34, 35-39, 40-44, 45-49, 50-54, 55-59.
5. inv-nodes: 0-2, 3-5, 6-8, 9-11, 12-14, 15-17, 18-20, 21-23, 24-26, 27-29, 30-32, 33-35, 36-39.
6. node-caps: yes, no.
7. deg-Malig: 1, 2, 3.
8. breast: left, right.
9. breast-quad: left-up, left-low, right-up, right-low, central.

◦ Missing Attribute Values: (denoted by "?")

Attribute #: Number of instances with missing values:

6. node-caps	8. breast
9. breast-quad	1. Class

◦ Class Distribution:

1. no-recurrence-events: 201 instances
2. recurrence-events: 85 instances

◦ Key Challenges and Usage:

-Preprocessing: Handling missing values, normalizing or standardizing data, and transforming data into a format suitable for machine learning algorithms.

-Classification Task: The primary task is to classify whether a given set of attributes indicates a benign or malignant tumor.

-Evaluation: Commonly evaluated using metrics like accuracy, precision, recall, and F1-score.

METHODOLOGY

DATA PRE-PROCESSING:

1. Importing Libraries:
pandas: Used for data manipulation and analysis.
train_test_split: A function to split data into training and testing sets.
StandardScaler: A preprocessing tool to standardize features by removing the mean and scaling to unit variance.

2. Standardizing the Features:
Label Encoding:
The le object refers to an instance of the LabelEncoder class from the sklearn.preprocessing module. LabelEncoder is used to convert categorical text data into numerical data. Each unique category is assigned a unique integer value.
Fitting and Transforming:
le.fit_transform(df['node']) performs two operations:
fit: This method identifies the unique categories in the df['node'] column and assigns a unique integer to each category.
transform: This method converts the categories into their corresponding numerical values based on the mapping established during the fitting.
Assigning Transformed Data:
df['node'] = le.fit_transform(df['node']):
The transformed numerical values replace the original categorical values in the node column of the DataFrame df.
Do it for all the columns.

VISUALIZATION:

For the data visualization part, it has used Seaborn as a library for

- count plot to check the distribution of breast cancer in specific age groups
- count plot to count the number of breast cancer in menopause
- count plot to count the number of breast cancer in specific tumor size
- count plot to identify the specific number with breast cancer in the left and right breast respectively
- count plot to identify the number of breast cancer in specific breast quadrants
- checking the distribution plot of all the columns
- creating a loop for getting the distribution plot for all the column
- creating a pair plot for the dataset
- constructing a heatmap for the correlation matrix
- boxplot for outliers' detection in the dataset

XGBOOST:

1.Importing Libraries:

XGBClassifier from XGBoost for building the model.

train_test_split from scikit-learn for splitting the dataset.

accuracy_score, classification_report, and confusion_matrix from scikit-learn for model evaluation.

Matplotlib and Seaborn for visualizing the confusion matrix.

2.Features

(X) are created by dropping the 'breast-quad' column from the dataset.

Target variable (y) is set to the 'breast-quad' column.

3.Data Splitting:

The dataset is split into training and testing sets using an 80-20 split.

The test_size parameter is set to 4.

4.Model Training:

An instance of XGBClassifier is created.

The model is trained on the training data using the fit method.

5.Model Evaluation:

The accuracy of the model is calculated and printed.

The classification report is generated, providing detailed metrics for each class's precision, recall, and F1 score.

The confusion matrix is printed, showing the number of correct and incorrect predictions.

6. Visualization:

The confusion matrix is visualized using a heatmap with Seaborn.

The heatmap provides a clear visual representation of the model's performance, highlighting areas of correct and incorrect predictions.

RANDOM FOREST:

1. Importing Libraries:

RandomForestClassifier from scikit-learn for building the model.

pandas and numpy for data manipulation and numerical operations.

train_test_split from scikit-learn for splitting the dataset.

confusion_matrix from scikit-learn for model evaluation.

2. Setting Random Seed:

A random seed is set using np.random.seed(0) to ensure reproducibility of the results.

3.Loading Data:

The dataset is assumed to be preloaded into a DataFrame named df.

The 'breast-quad' column is converted to a categorical type using pd.Categorical.

4. Splitting Data:

An 'is_train' column is added to the DataFrame to indicate whether a row will be used for training or testing. Rows with values less than or equal to 0.75 in the 'is_train' column are assigned to the training set, while the rest are assigned to the test set.

5. Creating Train and Test Sets:

The DataFrame is split into train and test sets based on the 'is_train' column.

The number of observations in each set is printed.

6. Feature Selection:

A list of feature column names is created, which includes the first four columns of the DataFrame.

The features are printed for verification.

7. Target Variable Conversion:

The target variable ('breast-quad') in the training set is converted into numerical values using pd.factorize.

8. Model Training:

An instance of RandomForestClassifier is created with n_jobs=2 for parallel processing and random_state=0 for reproducibility.

The classifier is trained using the training data (train[features]) and the numerical target variable (y).

9. Making Predictions:

Predictions are made on the test set using the trained classifier.

Artificial Neural Network (ANN):

1. Library Imports:

The necessary libraries for building and evaluating the ANN are imported, such as numpy, tensorflow, and keras.

2. Data Preparation:

The data is preprocessed and prepared for training the neural network. This typically involves normalizing the data and splitting it into training and testing sets.

3. Building the ANN Model:

The model is defined using the Sequential class from Keras.

Layers are added to the model using the Dense class, which includes specifying the number of neurons, activation functions, and input shape for the first layer.

3. Compiling the Model:

The model is compiled with a loss function (e.g., categorical_crossentropy for multi-class classification), an optimizer (e.g., adam), and metrics to monitor during training (e.g., accuracy).

4. Training the Model:

The model is trained using the fit method, where the training data and corresponding labels are provided. Validation data can also be specified to monitor the model's performance on unseen data during training.

5. Evaluating the Model:

After training, the model's performance is evaluated on the test set using the evaluate method.
Predictions are made using the predict method, and these predictions are compared to the true labels to generate evaluation metrics such as accuracy, precision, recall, and F1-score.

6. Results and Visualization:

The classification report and confusion matrix are generated to provide a detailed overview of the model's performance.
Visualizations, such as plotting the confusion matrix, help in understanding where the model performs well and where it might need improvement.

RESULT AND DISCUSSION

XGBOOST:

Classification Report

The classification report provides detailed metrics for each class in the dataset:

Precision: The ratio of correctly predicted positive observations to the total predicted positives.

Recall: The ratio of correctly predicted positive observations to the all observations in the actual class.

F1-score: The weighted average of Precision and Recall.

Support: The number of actual occurrences of the class in the test set.

For each class (1, 2, and 4):

Class 1,2,4:

Precision: 1.00,1.00,1.00 (100% of the instances predicted as class 1 were actually class 1)

Recall: 1.00,1.00,1.00 (100% of the actual class 1 instances were correctly identified)

F1-score: 1.00,1.00,1.00 (harmonic mean of precision and recall)

Support: 2,1,1 instances

Overall (accuracy): 100% accuracy is achieved across all instances

The confusion matrix shows the performance of the classification model on the test data.

[[2 0 0]

[0 1 0]

[0 0 1]]

The rows represent the true classes, while the columns represent the predicted classes.

RANDOM FOREST:

The array you provided represents a sequence of class labels encoded as integers. Each element in the array corresponds to a specific class label for an instance in a dataset. The dtype=int64 indicates that the array elements are 64-bit integers.

Here's a breakdown of what the values might represent in the context of classification:

Classes: The integers in the array represent different class labels. For example:

0 might correspond to one class (e.g., class 0)
1 might correspond to another class (e.g., class 1)
2 might correspond to yet another class (e.g., class 2)
3 might correspond to another class (e.g., class 3)
4 might correspond to yet another class (e.g., class 4)
Class Distribution: By examining the array, you can get a sense of how many instances belong to each class. This can help understand the class distribution in your dataset. For example, counting the occurrences of each number will tell you how many instances belong to each class

ANN:

Classification Report:
The classification report provides the performance metrics of the model:

Precision: The ratio of correctly predicted positive observations to the total predicted positives.
Recall: The ratio of correctly predicted positive observations to the all observations in the actual class.
F1-score: The weighted average of precision and recall. It ranges from 0 to 1, with 1 being the best possible score.
Support: The number of actual occurrences of each class in the y_test data.
Here is the breakdown for each class (0 through 4):

Class 0,1,2,3,4:

Precision: 0.00,0.32,040,0.06,0.00 (none of the predicted instances are true positives)
Recall: 0.00,0.26,0.33,0.33,0.00 (none of the actual instances are predicted correctly)
F1-score: 0.00,0.29,0.36,0.11,0.00
Support: 8,23,18,3,6 (there are 8 instances of class 0 in the test set)

Overall Metrics:

Accuracy: 0.22 (The model correctly predicts 22% of the instances).
Macro average (macro avg):
Precision: 0.16
Recall: 0.19
F1-score: 0.15
Weighted average (weighted avg):
Precision: 0.25
Recall: 0.22
F1-score: 0.23

GROUP CONTRIBUTION:

Abhishek Jha- Deployment
Ashish kumar - Visualisation; Random Forest
Ayushi Singh - Pre-processing, ANN
Ayushi Vardhan - XgBoost; Report

CONCLUSION

Potential Future Work

Data Preprocessing:
Feature Engineering: Investigate additional feature engineering techniques to improve the model's performance.
Handling Class Imbalance: Use techniques such as oversampling, undersampling, or synthetic data generation (e.g., SMOTE) to handle class imbalances.

Model Improvement:
Hyperparameter Tuning: Perform hyperparameter tuning using techniques like Grid Search or Random Search to find the best parameters for the RandomForestClassifier.

Model Ensemble: Explore ensemble methods such as boosting (e.g., Gradient Boosting, XGBoost) or stacking multiple models to improve performance.

Evaluation Metrics:
Cross-Validation: Implement cross-validation to ensure that the model's performance is consistent across different subsets of the data.
Confusion Matrix Analysis: Analyze confusion matrices to understand where the model is making the most mistakes.

Algorithm Exploration:
Try Different Algorithms: Experiment with other classification algorithms such as Support Vector Machines (SVM), k-Nearest Neighbors (k-NN), or neural networks to see if they perform better on the dataset.

Feature Importance:
Analyze Feature Importance: Use the feature importances provided by the RandomForestClassifier to understand which features are most influential in making predictions and potentially refine the feature set.