Gout Predictions

# Introduction:

**Overview of Gout**

Gout is a type of inflammatory arthritis characterized by sudden, severe attacks of pain, redness, swelling, and tenderness in the joints, most commonly the base of the big toe. It is caused by the accumulation of urate crystals in the joints, which leads to inflammation and pain. Gout typically affects men more often than women and is associated with various risk factors including diet, genetics, obesity, and certain medical conditions such as hypertension and diabetes.

The primary goal of managing gout is to reduce pain and inflammation, prevent future gout attacks, and lower uric acid levels in the blood through lifestyle changes and medication.

**Role of Machine Learning in Gout Prediction**

Machine learning (ML) can play a crucial role in predicting the occurrence of gout in individuals by analyzing various factors that contribute to its development. Here's how ML can aid in gout prediction:

1. Identifying Risk Factors: Machine learning algorithms can analyze large datasets containing demographic information, medical history, lifestyle factors, and biomarkers to identify patterns and correlations associated with gout development. By understanding these risk factors, healthcare providers can better assess an individual's likelihood of developing gout.
2. Predictive Modeling: ML techniques such as classification algorithms can be trained on historical data to build predictive models for gout occurrence. These models can consider a wide range of features and variables to generate personalized risk scores or probabilities for individuals. By leveraging predictive modeling, healthcare providers can proactively identify individuals at high risk of developing gout and implement preventive measures or interventions.
3. Early Detection and Intervention: ML models can assist in early detection of gout by analyzing subtle changes in patient data and symptoms that may precede a gout attack. Early detection allows for timely intervention and treatment, potentially reducing the severity and frequency of gout attacks and minimizing long-term joint damage.
4. Personalized Treatment Plans: ML algorithms can analyze patient data to generate personalized treatment plans tailored to individual risk profiles, comorbidities, and preferences. By optimizing treatment strategies based on patient-specific characteristics, healthcare providers can improve outcomes and quality of life for individuals with gout.

Overall, machine learning holds promise in advancing the field of gout prediction by leveraging data-driven approaches to better understand, predict, and manage this debilitating condition.

# Data Acquisition:

1. *Data Source Selection*: For the Gout Prediction the data source is considered from physionet open-source data. <https://physionet.org/content/emer-complaint-gout/1.0/>
2. *Data Upload to Redshift*: The downloaded data from the physionet is been uploaded to the Redshift environment.
3. *Data Preprocessing*: Clean the raw data by removing duplicates, handling special characters, and ensuring data integrity before further analysis.

# Exploratory Data Analysis (EDA):

1. *Data Exploration*: Use pandas and other Python libraries to load the dataset and gain an initial understanding of its structure.
2. *Descriptive Statistics*: Calculate descriptive statistics for numerical features (e.g., mean, median, standard deviation) and categorical features (e.g., count, unique values).
3. *Visualization*: Create visualizations (e.g., histograms, bar plots, scatter plots) to explore the distribution of target classes and feature relationships. Identify outliers, trends, and patterns that may influence model development.
4. *Key Features and Insights*: The data class is skewed, and class N are high as compared to Y or U. The distribution is 97%, 2% & 1% respectively.

# Data Cleaning:

1. Handling Missing Values: Use techniques such as imputation (e.g., mean, median, mode) or deletion (e.g., dropping rows or columns) to handle missing data appropriately.
2. Handling Imbalanced Classes: Apply techniques like Synthetic Minority Over-sampling Technique (SMOTE) to address class imbalance by generating synthetic samples for minority classes.
3. Feature Engineering: Since only 1 column is utilized, TFIDF is performed on the dataset.

# Model Building:

1. *Train-Test Split*: Split the dataset into training and testing sets (e.g., 70-30 or 80-20 split) to evaluate model performance on unseen data.
2. *Initial Model Training*: Train a baseline model (e.g., Random Forest classifier) without applying SMOTE to establish a performance baseline.
3. *Evaluation Metrics*: Evaluate model performance using appropriate evaluation metrics (e.g., precision, recall, F1-score, accuracy) to assess classification performance.
4. *Model Improvement*: Address class imbalance using techniques like SMOTE and retrain the model to evaluate performance improvements.
5. *Hyperparameter* Tuning: Optimize model hyperparameters using techniques like grid search or random search to maximize performance.

# Evaluation:

1. Initial Model Training:

Random Forest Classifier (Without SMOTE):

|  |  |  |
| --- | --- | --- |
| **Class** | **Precision** | **Recall** |
| **N class** | 0.97 | 1 |
| **U class** | 0 | 0 |
| **Y class** | 0 | 0 |
| **Accuracy** | 0.97 | |

1. Model Improvement: Random Forest Classifier (With SMOTE)

|  |  |  |
| --- | --- | --- |
| **Class** | **Precision** | **Recall** |
| **N class** | 0.97 | 1 |
| **U class** | 0.5 | 0.03 |
| **Y class** | 0.86 | 0.14 |
| **Accuracy** | 0.97 | |

1. Additional Models:

XGBoost and Support Vector Classifier (SVC) were tried with varying performance metrics.

SVC (After Hyperparameter Tuning):

Best Hyperparameters: {'C': 1, 'kernel': 'rbf'}

Performance metrics comparable to or better than other models.

# Model Serialization:

1. Pipeline Creation:

Build a machine learning pipeline that includes data preprocessing, feature engineering, and model training steps to ensure reproducibility.

1. Model Serialization:

Serialize the trained model and preprocessing steps using pickle or joblib to save the model's state for future deployment and inference.

# Model Deployment:

1. *Deployment Environment*: Deploy the trained model in a cloud environment Amazon SageMaker for scalable and reliable model hosting.
2. *Endpoint Creation:*  Create an endpoint in Amazon SageMaker to expose the deployed model for real-time predictions.
3. *Inference Script:* Develop an inference script (`inference.py`) that handles incoming HTTP requests, performs data preprocessing, and returns predictions using the deployed model.
4. *Testing and Monitoring:*  Test the deployed endpoint with sample data to ensure functionality and performance. Set up monitoring and logging to track model performance metrics, identify anomalies, and troubleshoot issues in real-time.

# Folder Structure:

1. Data: The downloaded data is present in this folder.
2. Model: All the models that are developed along with the pipeline and archive files are stored here.
3. Files to consider: To explore the files follow the files below:
   1. Gout\_Prediction.ipynb: Gout Prediction in the SageMaker environment. This include the Database calls, Preprocessing, Model Building and Deployment as a endpoint.
   2. Push Data to Redshift.ipynb: This file was created to push the downloaed files to Redshift.
   3. direct\_predict\_gout.py: This file is created to generate and predict the Gout directly from the NB without Sagemaker intervention.
   4. host\_gout.py: This is to trigger the streamlit application for Gout Prediction.
   5. inference.py: File for Sagemaker backend to create end point.
   6. predict\_and\_decode.py: Class code file to predict the gout.