

Predicting Doctor Prescriptions Using Machine Learning



**Team:**

Menna Tarek

Salma Ahmed

Mahmoud Tag

Mahmoud Atef

Omar Ali

**Introduction**

This project aims to predict whether a doctor will prescribe medication based on interactions with a medical representative. The dataset contains features such as doctor specialization, clinic type, and medication type. We will clean the data, visualize patterns, preprocess features, and build several machine learning models to make predictions.

**1. Join Two Tables Using SQL**

The dataset is stored in two tables: one for doctor details and another for medical representative interactions. We perform an SQL join to combine them based on a common key (e.g., doctor\_id).

sql

Copy code

SELECT doctors.\*, interactions.\*

FROM doctors

JOIN interactions

ON doctors.doctor\_id = interactions.doctor\_id;

This query combines both tables into one dataset that we can then use for further analysis.

**2. Data Cleaning**

**2.1 Check for Missing Values**

We inspect the dataset for missing values to ensure there are no incomplete records that could affect the model’s performance.

python

Copy code

# Check for missing values

df.isnull().sum()

If missing values are found, we can either remove the rows or impute missing data using methods like the mean, median, or mode depending on the feature type.

**2.2 Check for Skewness**

We check the skewness of numerical features to ensure that the distribution is appropriate for model training.

python

Copy code

# Check skewness of numerical columns

df.skew()

**2.3 Handle Skewness and Outliers**

To address skewness, we can apply transformations like the logarithmic transformation. Outliers can be identified using box plots and treated by capping or removing extreme values.

python

Copy code

# Handle skewness with log transformation (if necessary)

df['feature'] = np.log1p(df['feature'])

# Remove or cap outliers

df = df[df['feature'] < df['feature'].quantile(0.99)]

**3. Data Visualization**

**3.1 Df.info**

We use the .info() function to get a concise summary of the dataset, which includes data types, non-null counts, and memory usage.

python

Copy code

df.info()

**3.2 Df.describe**

We generate summary statistics for numerical columns using .describe(), which provides information such as the mean, median, and quartiles.

python

Copy code

df.describe()

**3.3 Histogram for Numerical Features**

We plot histograms to visualize the distribution of numerical features.

python

Copy code

df.hist(figsize=(10, 8))

plt.show()

**3.4 Bar Plot for Categorical Features**

Bar plots are used to visualize the distribution of categorical variables.

python

Copy code

df['categorical\_feature'].value\_counts().plot(kind='bar')

plt.show()

**3.5 Box Plot to Identify Outliers and Skewness**

Box plots help in identifying outliers and detecting skewness in the data.

python

Copy code

sns.boxplot(data=df['numerical\_feature'])

plt.show()

**4. Data Preprocessing**

**4.1 Encode Categorical Features**

Categorical features are encoded into numerical values using one-hot encoding or label encoding to make them usable by machine learning algorithms.

python

Copy code

# One-hot encoding for categorical features

df = pd.get\_dummies(df, columns=['categorical\_feature'], drop\_first=True)

**4.2 Split Data into Training and Testing Sets**

The dataset is split into training and testing sets to evaluate the model’s performance on unseen data.

python

Copy code

from sklearn.model\_selection import train\_test\_split

X = df.drop('target', axis=1)

y = df['target']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**4.3 Standardize Numerical Features**

We standardize numerical features to ensure that they have a mean of 0 and a standard deviation of 1, using StandardScaler.

python

Copy code

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

**5. Model Selection and Training**

We train several machine learning models and perform hyperparameter tuning to select the best-performing model.

**5.1 Models**

The following machine learning models are used for training:

* Logistic Regression
* Random Forest
* Support Vector Machine (SVM)
* XGBoost
* AdaBoost
* Gradient Boosting
* Bagging

python

Copy code

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier

from sklearn.svm import SVC

from xgboost import XGBClassifier

# Initialize models

log\_reg = LogisticRegression()

rf = RandomForestClassifier()

svm = SVC()

xgb = XGBClassifier()

ada = AdaBoostClassifier()

gb = GradientBoostingClassifier()

# Train models

log\_reg.fit(X\_train, y\_train)

rf.fit(X\_train, y\_train)

svm.fit(X\_train, y\_train)

xgb.fit(X\_train, y\_train)

ada.fit(X\_train, y\_train)

gb.fit(X\_train, y\_train)

**5.2 Hyperparameter Tuning**

We use GridSearchCV and RandomizedSearchCV for hyperparameter tuning to find the optimal parameters for each model.

python

Copy code

from sklearn.model\_selection import GridSearchCV

# Example grid search for Random Forest

param\_grid = {

'n\_estimators': [100, 200],

'max\_depth': [10, 20]

}

grid\_search = GridSearchCV(estimator=rf, param\_grid=param\_grid, cv=3)

grid\_search.fit(X\_train, y\_train)

**5.3 Model Evaluation**

After training, we evaluate model performance using accuracy, precision, recall, F1 score, and the confusion matrix.

python

Copy code

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, classification\_report, confusion\_matrix

# Model evaluation

y\_pred = rf.predict(X\_test)

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred)}")

print(f"Precision: {precision\_score(y\_test, y\_pred)}")

print(f"Recall: {recall\_score(y\_test, y\_pred)}")

print(f"F1 Score: {f1\_score(y\_test, y\_pred)}")

print(classification\_report(y\_test, y\_pred))

print(confusion\_matrix(y\_test, y\_pred))

**5.4 Classification Report and Confusion Matrix**

The classification report provides a detailed analysis of precision, recall, and F1 score. The confusion matrix shows the number of true positives, true negatives, false positives, and false negatives.

**Conclusion**

In this project, we built several machine learning models to predict whether a doctor will prescribe a medication based on historical interactions with medical representatives. After cleaning, visualizing, and preprocessing the data, we trained models such as Logistic Regression, Random Forest, SVM, and others. Hyperparameter tuning allowed us to select the best-performing model. The final evaluation showed that the Random Forest model provided the highest accuracy and a well-balanced precision-recall score.

This predictive model can assist pharmaceutical companies in optimizing their strategies for medical representative interactions, leading to more efficient operations and better healthcare outcomes