untitled40

August 7, 2023

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import StandardScaler, LabelEncoder
```

[2]:
```
df = pd.read_csv("C:/Users/DELL/Downloads/Dataset.csv")
df.head()
```

[2]:
```
   encounter_id  patient_id  hospital_id  hospital_death   age    bmi  \
0         66154       25312          118               0  68.0  22.73
1        114252       59342           81               0  77.0  27.42
2        119783       50777          118               0  25.0  31.95
3         79267       46918          118               0  81.0  22.64
4         92056       34377           33               0  19.0    NaN

   elective_surgery  ethnicity gender  height  … aids  cirrhosis  \
0                 0  Caucasian      M   180.3  …  0.0        0.0
1                 0  Caucasian      F   160.0  …  0.0        0.0
2                 0  Caucasian      F   172.7  …  0.0        0.0
3                 1  Caucasian      F   165.1  …  0.0        0.0
4                 0  Caucasian      M   188.0  …  0.0        0.0

   diabetes_mellitus hepatic_failure immunosuppression  leukemia  lymphoma  \
0                1.0             0.0               0.0       0.0       0.0
1                1.0             0.0               0.0       0.0       0.0
2                0.0             0.0               0.0       0.0       0.0
3                0.0             0.0               0.0       0.0       0.0
4                0.0             0.0               0.0       0.0       0.0

   solid_tumor_with_metastasis  apache_3j_bodysystem  apache_2_bodysystem
0                          0.0                Sepsis       Cardiovascular
1                          0.0           Respiratory          Respiratory
2                          0.0             Metabolic            Metabolic
3                          0.0        Cardiovascular       Cardiovascular
```

```
4                           0.0                 Trauma                Trauma

[5 rows x 186 columns]
```

[3]:

```
df.dtypes
```

[3]:

```
encounter_id                     int64
patient_id                       int64
hospital_id                      int64
hospital_death                   int64
age                            float64
                                  …
leukemia                       float64
lymphoma                       float64
solid_tumor_with_metastasis    float64
apache_3j_bodysystem            object
apache_2_bodysystem             object
Length: 186, dtype: object
```

[4]:

```
df.columns
```

[4]:

```
Index(['encounter_id', 'patient_id', 'hospital_id', 'hospital_death', 'age',
       'bmi', 'elective_surgery', 'ethnicity', 'gender', 'height',
       …
       'aids', 'cirrhosis', 'diabetes_mellitus', 'hepatic_failure',
       'immunosuppression', 'leukemia', 'lymphoma',
       'solid_tumor_with_metastasis', 'apache_3j_bodysystem',
       'apache_2_bodysystem'],
      dtype='object', length=186)
```

[5]:

```
# Handle Missing Values
# For numerical features, fill missing values with the mean
numeric_features = ['age', 'bmi']
df[numeric_features] = df[numeric_features].fillna(df[numeric_features].mean())
```

[6]:

```
# Encode Categorical Features
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import StandardScaler, LabelEncoder

encoder = LabelEncoder()
categorical_features = ['ethnicity', 'gender']
for col in categorical_features:
    df[col] = encoder.fit_transform(df[col].astype(str))
```

[23]:

```
# Interactive Visualizations using Plotly Express

# Scatter plot to explore relationships between age and BMI
scatter_age_bmi = px.scatter(df, x='age', y='bmi', color='sepsis_label',
                             title='Relationship between Age, BMI, and Sepsis')
scatter_age_bmi.show()
```

[25]:

```
# Parallel coordinates plot to visualize multiple features at once
parallel_plot = px.parallel_coordinates(df, dimensions=['age', 'bmi',␣
 ↪'ethnicity', 'gender'],
                                        color='sepsis_label',
                                        labels={'age': 'Age', 'bmi': 'BMI',␣
 ↪'ethnicity': 'Ethnicity', 'gender': 'Gender'},
                                        title='Parallel Coordinates Plot of␣
 ↪Features and Sepsis')
parallel_plot.show()
```

[27]:

```
# Box plot to compare the distribution of age among sepsis and non-sepsis cases
box_age = px.box(df, x='sepsis_label', y='age', title='Age Distribution among␣
 ↪Sepsis and Non-Sepsis Cases')
box_age.show()
```

[28]:

```
# Histogram to explore the distribution of BMI among sepsis and non-sepsis cases
hist_bmi = px.histogram(df, x='bmi', color='sepsis_label', title='BMI␣
 ↪Distribution among Sepsis and Non-Sepsis Cases')
hist_bmi.show()
```

[8]:

```
# Define sepsis label based on diagnoses or other relevant indicators
df['sepsis_label'] = (df['aids'] == 1) | (df['cirrhosis'] == 1) |␣
 ↪(df['diabetes_mellitus'] == 1)
```

[9]:

```
# Select relevant features for sepsis detection
selected_features = ['age', 'bmi', 'sepsis_label']  # Update with the correct␣
 ↪feature names
data_selected = df[selected_features]
```

[10]:

```
# Splitting data into features (X) and target (y)
X = data_selected.drop(['sepsis_label'], axis=1)
y = data_selected['sepsis_label']
```

[11]:

```
# Normalize Numerical Features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Splitting data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[12]:

```
# Train and Evaluate Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print("Random Forest Accuracy:", rf_accuracy)
print("Random Forest Classification Report:\n", classification_report(y_test,␣
 ↪rf_predictions))
```

```
Random Forest Accuracy: 0.6654854712969526
Random Forest Classification Report:
               precision    recall  f1-score   support

       False       0.77      0.80      0.78     13987
        True       0.27      0.24      0.26      4356

    accuracy                           0.67     18343
   macro avg       0.52      0.52      0.52     18343
weighted avg       0.65      0.67      0.66     18343
```

[13]:

```
# Train and Evaluate Support Vector Machine (SVM)
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)
svm_predictions = svm_model.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_predictions)
print("SVM Accuracy:", svm_accuracy)
print("SVM Classification Report:\n", classification_report(y_test,␣
 ↪svm_predictions))
```

```
SVM Accuracy: 0.7625252139780843
SVM Classification Report:
               precision    recall  f1-score   support

       False       0.76      1.00      0.87     13987
        True       0.00      0.00      0.00      4356

    accuracy                           0.76     18343
   macro avg       0.38      0.50      0.43     18343
weighted avg       0.58      0.76      0.66     18343


C:\Users\DELL\anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
```

```
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\DELL\anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\DELL\anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[14]:

```
# Train and Evaluate K-Nearest Neighbors (KNN)
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
knn_predictions = knn_model.predict(X_test)
knn_accuracy = accuracy_score(y_test, knn_predictions)
print("KNN Accuracy:", knn_accuracy)
print("KNN Classification Report:\n", classification_report(y_test,␣
 ↪knn_predictions))
```

```
KNN Accuracy: 0.7220192989151175
KNN Classification Report:
               precision    recall  f1-score   support

       False       0.77      0.90      0.83     13987
        True       0.32      0.15      0.20      4356

    accuracy                           0.72     18343
   macro avg       0.54      0.52      0.52     18343
weighted avg       0.66      0.72      0.68     18343
```

[21]:

```
# Create a DataFrame to store model names and accuracies
import plotly.express as px
import numpy as np

models = ['Random Forest', 'SVM', 'KNN']
accuracies = [rf_accuracy, svm_accuracy, knn_accuracy]
std_devs = [np.std(rf_predictions), np.std(svm_predictions), np.
 ↪std(knn_predictions)]
model_data = pd.DataFrame({'Model': models, 'Accuracy': accuracies, 'Std Dev':␣
 ↪std_devs})

# Plot Advanced-Level Graph using Plotly
fig = px.bar(model_data, x='Model', y='Accuracy', error_y='Std Dev',
             labels={'Accuracy': 'Accuracy Score', 'Std Dev': 'Standard␣
 ↪Deviation'},
             title='Model Accuracy Comparison',␣
 ↪color_discrete_sequence=['royalblue'])
fig.update_layout(xaxis={'categoryorder': 'total descending'})
fig.show()
```

[35]:

```
import time  # For simulating new data arrival

# Online Learning Simulation

for _ in range(10):  # Simulate 10 new data points
    # Simulate new data point
    new_data_point = {
        'age': np.random.randint(18, 90),
        'bmi': np.random.uniform(15, 40),
        'ethnicity': np.random.choice(data['ethnicity'].unique()),
        'gender': np.random.choice(data['gender'].unique()),
        'sepsis_label': np.random.choice([True, False])
    }


    # Update models with the new data point
    new_data_df = pd.DataFrame([new_data_point])
    X_new = scaler.transform(new_data_df[['age', 'bmi']])
    y_new = new_data_df['sepsis_label'].values

    # Ensure that both classes are represented in the data
    if len(np.unique(y_new)) == 2:
        # Update RandomForest model with warm start
        rf_model.n_estimators += 1
        rf_model.fit(X_new, y_new)

        # Update SVM and KNN models with the new data point
        svm_model.fit(X_new, y_new)
        knn_model.fit(X_new, y_new)

    # Print updated model accuracies
    print("Random Forest Accuracy:", rf_model.score(X_test, y_test))
    print("SVM Accuracy:", svm_model.score(X_test, y_test))
    print("KNN Accuracy:", knn_model.score(X_test, y_test))

    # Pause to simulate data arrival interval
    time.sleep(3)  # Wait for 3 seconds before simulating the next data point
```

```
Random Forest Accuracy: 0.7625252139780843
SVM Accuracy: 0.7625252139780843
```

```
KNN Accuracy: 0.7220192989151175
Random Forest Accuracy: 0.7625252139780843
SVM Accuracy: 0.7625252139780843
KNN Accuracy: 0.7220192989151175
Random Forest Accuracy: 0.7625252139780843
SVM Accuracy: 0.7625252139780843
KNN Accuracy: 0.7220192989151175
Random Forest Accuracy: 0.7625252139780843
SVM Accuracy: 0.7625252139780843
KNN Accuracy: 0.7220192989151175
Random Forest Accuracy: 0.7625252139780843
SVM Accuracy: 0.7625252139780843
KNN Accuracy: 0.7220192989151175
Random Forest Accuracy: 0.7625252139780843
SVM Accuracy: 0.7625252139780843
KNN Accuracy: 0.7220192989151175
Random Forest Accuracy: 0.7625252139780843
SVM Accuracy: 0.7625252139780843
KNN Accuracy: 0.7220192989151175
Random Forest Accuracy: 0.7625252139780843
SVM Accuracy: 0.7625252139780843
KNN Accuracy: 0.7220192989151175
Random Forest Accuracy: 0.7625252139780843
SVM Accuracy: 0.7625252139780843
KNN Accuracy: 0.7220192989151175
Random Forest Accuracy: 0.7625252139780843
SVM Accuracy: 0.7625252139780843
KNN Accuracy: 0.7220192989151175
```

[41]:

```python
# Perform k-fold cross-validation
from sklearn.model_selection import cross_val_score

k = 5  # Number of folds
cv_scores = cross_val_score(rf_model, X, y, cv=k, scoring='accuracy')

# Print cross-validation scores for each fold
for fold, score in enumerate(cv_scores, start=1):
    print(f"Fold {fold}: Accuracy = {score:.4f}")

# Calculate and print the mean and standard deviation of cross-validation scores
mean_score = np.mean(cv_scores)
std_dev_score = np.std(cv_scores)
print(f"Mean Accuracy: {mean_score:.4f}")
print(f"Standard Deviation: {std_dev_score:.4f}")
```

```
Fold 1: Accuracy = 0.6731
Fold 2: Accuracy = 0.6670
Fold 3: Accuracy = 0.6691
Fold 4: Accuracy = 0.6671
Fold 5: Accuracy = 0.6727
Mean Accuracy: 0.6698
Standard Deviation: 0.0026
```

[51]:

```
# Perform k-fold cross-validation for the SVM model
svm_model = SVC(kernel='linear', random_state=42)
k = 5  # Number of folds
svm_cv_scores = cross_val_score(svm_model, X, y, cv=k, scoring='accuracy')
```

[52]:

```
# Perform k-fold cross-validation for the KNN model
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_cv_scores = cross_val_score(knn_model, X, y, cv=k, scoring='accuracy')
```

[53]:

```
# Calculate and print the mean and standard deviation of cross-validation␣
 ↪scores for SVM
svm_mean_score = np.mean(svm_cv_scores)
svm_std_dev_score = np.std(svm_cv_scores)
print("SVM:")
print(f"Mean Accuracy: {svm_mean_score:.4f}")
print(f"Standard Deviation: {svm_std_dev_score:.4f}")

# Calculate and print the mean and standard deviation of cross-validation␣
 ↪scores for KNN
knn_mean_score = np.mean(knn_cv_scores)
knn_std_dev_score = np.std(knn_cv_scores)
print("KNN:")
print(f"Mean Accuracy: {knn_mean_score:.4f}")
print(f"Standard Deviation: {knn_std_dev_score:.4f}")
```

```
SVM:
Mean Accuracy: 0.7645
Standard Deviation: 0.0000
KNN:
Mean Accuracy: 0.7238
Standard Deviation: 0.0025
```

[44]:

```
# Initialize individual models
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
svm_model = SVC(kernel='linear', probability=True, random_state=42)
knn_model = KNeighborsClassifier(n_neighbors=5)
```

[47]:

```
# Initialize the VotingClassifier with the individual models
ensemble_model = VotingClassifier(
    estimators=[('rf', rf_model), ('svm', svm_model), ('knn', knn_model)],
    voting='soft'  # Use 'soft' voting for probability-based predictions
)
```

[48]:

```
# Train the ensemble model
ensemble_model.fit(X_train, y_train)
```

[48]:

```
VotingClassifier(estimators=[('rf', RandomForestClassifier(random_state=42)),
                             ('svm',
                              SVC(kernel='linear', probability=True,
                                  random_state=42)),
                             ('knn', KNeighborsClassifier())],
                 voting='soft')
```

[49]:

```
# Evaluate the ensemble model
ensemble_accuracy = ensemble_model.score(X_test, y_test)
print("Ensemble Model Accuracy:", ensemble_accuracy)
```

```
Ensemble Model Accuracy: 0.7397917461702012
```