HI KNN.

October 8, 2024

1 K-Nearest Neighbors: Health Insurance

1.1 Load Data

1.2 Training Phase

[ ]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import  accuracy_score, confusion_matrix,␣
 ↪classification_report, roc_curve, auc
import matplotlib.pyplot as plt

# Load the dataset
numeric_df = pd.read_csv('cleaned_data.csv')

# Prepare feature matrix and target vector
X = numeric_df.drop('DentalPlan', axis=1)
y = numeric_df['DentalPlan']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define the parameter grid
param_grid = {
    'n_neighbors': np.arange(1, 21),
    'weights': ['uniform', 'distance']
}

# Initialize the KNN classifier
```

```
knn = KNeighborsClassifier()

# Perform grid search with cross-validation
grid_search = GridSearchCV(knn, param_grid, cv=5)
grid_search.fit(X_train_scaled, y_train)

# Get the best parameters
best_params = grid_search.best_params_
best_k = best_params['n_neighbors']
best_weights = best_params['weights']
print(f"Best number of neighbors: {best_k}, Best weights: {best_weights}")

# Train the model with the best parameters
knn_best = KNeighborsClassifier(n_neighbors=best_k, weights=best_weights)
knn_best.fit(X_train_scaled, y_train)

# Make predictions
y_pred = knn_best.predict(X_test_scaled)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

1.3 ROC Curve

[ ]:
```
y_prob = knn_best.predict_proba(X_test_scaled)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='red', lw=2, label='ROC curve (area = {:.2f})'.
  ↪format(roc_auc))
plt.plot([0, 1], [0, 1], color='blue', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()

print(f"AUC: {roc_auc:.2f}")
```

1.4 Heat Map: Hyperparameter Learning: n_neighbors and weight

[ ]:
```
import seaborn as sns
results = pd.DataFrame(grid_search.cv_results_)

pivot_table = results.pivot_table(index='param_weights',␣
 ↪columns='param_n_neighbors', values='mean_test_score')

# Plotting
plt.figure(figsize=(10, 6))
sns.heatmap(pivot_table, annot=True, fmt=".2f", cmap="YlGnBu",␣
 ↪cbar_kws={'label': 'Mean Test Score'})
plt.title('Hyperparameter Tuning: Health Insurance: KNN Weights vs.␣
 ↪n_neighbors')
plt.xlabel('Number of Neighbors')
plt.ylabel('Weights')
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.ylim(len(pivot_table) - 0.05, -0.55)
plt.show()
```

[ ]:
```
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

1.5 Training Time

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

def evaluate_knn(X_train, y_train, X_test, y_test, k_values):
    train_accuracies = []
    test_accuracies = []

    for k in k_values:

        knn = KNeighborsClassifier(n_neighbors=k)

        # Fit the model on the training data
        knn.fit(X_train, y_train)

        # Predict on the training set
        y_train_pred = knn.predict(X_train)
        # Predict on the test set
        y_test_pred = knn.predict(X_test)
```

```
        # Calculate accuracy for training and test sets
        train_accuracy = accuracy_score(y_train, y_train_pred)
        test_accuracy = accuracy_score(y_test, y_test_pred)

        # Append accuracies to the lists
        train_accuracies.append(train_accuracy)
        test_accuracies.append(test_accuracy)

    return train_accuracies, test_accuracies

# Sample Data
X_train_sample = X_train_scaled[:1000]
y_train_sample = y_train[:1000]
X_test_sample = X_test_scaled[:300]
y_test_sample = y_test[:300]

# K values to evaluate
k_values = range(1, 15)

# Evaluate KNN
train_accuracies, test_accuracies = evaluate_knn(X_train_sample,␣
 ↪y_train_sample, X_test_sample, y_test_sample, k_values)

# Plotting the results
plt.figure(figsize=(10, 6))
plt.plot(k_values, train_accuracies, label='Training Accuracy', marker='o')
plt.plot(k_values, test_accuracies, label='Validation Accuracy', marker='o')
plt.title('KNN Hyperparameter Tuning Curve: Health Insurance')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.xticks(k_values)
plt.grid()
plt.legend()
plt.show()
```

1.6 Training and Validation Error

[ ]:
```
n_neighbors_range = range(1, 15)
train_scores = []
val_scores = []



# Iterate over the range of n_neighbors
for n_neighbors in n_neighbors_range:

    knn = KNeighborsClassifier(n_neighbors=n_neighbors, weights='distance')
```

```
    knn.fit(X_train, y_train)

    # Predict on training and validation sets
    y_train_pred = knn.predict(X_train)
    y_val_pred = knn.predict(X_test)

    # Calculate accuracy scores
    train_accuracy = accuracy_score(y_train, y_train_pred)
    val_accuracy = accuracy_score(y_test, y_val_pred)

    train_scores.append(train_accuracy)
    val_scores.append(val_accuracy)

# Plotting the training and validation error curves
plt.figure(figsize=(10, 6))
plt.plot(n_neighbors_range, 1 - np.array(train_scores), marker='o',␣
 ↪linestyle='-', color='blue', label='Training Error')
plt.plot(n_neighbors_range, 1 - np.array(val_scores), marker='o',␣
 ↪linestyle='-', color='orange', label='Validation Error')
plt.title('Training and Validation Error Curves for KNeighborsClassifier')
plt.xlabel('Number of Neighbors (n_neighbors)')
plt.ylabel('Error (1 - Accuracy)')
plt.xticks(n_neighbors_range)
plt.legend()
plt.grid(True)
plt.show()
```

1.7 Training Time

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import time

def evaluate_knn(X_train, y_train, X_test, y_test, k_values):
    train_accuracies = []
    test_accuracies = []
    training_times = []

    for k in k_values:

        knn = KNeighborsClassifier(n_neighbors=k)

        # Start timing
        start_time = time.time()
```

```
        # Fit the model on the training data
        knn.fit(X_train, y_train)

        # Stop timing
        training_time = time.time() - start_time

        # Predict on the training set
        y_train_pred = knn.predict(X_train)
        # Predict on the test set
        y_test_pred = knn.predict(X_test)

        # Calculate accuracy for training and test sets
        train_accuracy = accuracy_score(y_train, y_train_pred)
        test_accuracy = accuracy_score(y_test, y_test_pred)

        # Append accuracies and training time to the lists
        train_accuracies.append(train_accuracy)
        test_accuracies.append(test_accuracy)
        training_times.append(training_time)

    return train_accuracies, test_accuracies, training_times

# Sample Data
X_train_sample = X_train_scaled[:1000]
y_train_sample = y_train[:1000]
X_test_sample = X_test_scaled[:300]
y_test_sample = y_test[:300]

# K values to evaluate
k_values = range(1, 14)

# Evaluate KNN
train_accuracies, test_accuracies, training_times = evaluate_knn(
    X_train_sample, y_train_sample, X_test_sample, y_test_sample, k_values
)


# Plotting
plt.figure(figsize=(10, 6))
plt.plot(k_values, training_times, label='Training Time', marker='o',␣
 ↪color='orange')
plt.title('Training Time for Different k Values in KNN: Health Insurance')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Training Time (seconds)')
plt.xticks(k_values)
plt.grid()
```

```
plt.legend()
plt.show()
```

1.8 Cross Validation Scores Across Folds

[ ]:
```
import numpy as np
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Evaluate the model with cross-validation
best_model = grid_search.best_estimator_
cv_scores = cross_val_score(best_model, X_train, y_train, cv=5)
print("Cross-Validation Scores:", cv_scores)
print("Mean Cross-Validation Score:", np.mean(cv_scores))
```

[ ]:
```
cv_scores = [0.9758502,  0.97523702, 0.97495401, 0.97561436, 0.97542453]
mean_cv_score = 0.9754160221171484


plt.figure(figsize=(10, 6))
plt.plot(range(1, len(cv_scores) + 1), cv_scores, marker='o', linestyle='-',␣
 ↪color='blue', label='Cross-Validation Scores')
plt.axhline(y=mean_cv_score, color='red', linestyle='--', label='Mean CV Score')
plt.title('Cross-Validation Scores Across Folds: Health Insurance')
plt.xlabel('Fold Number')
plt.ylabel('Accuracy Score')
plt.ylim(0.9725, 0.9775)
plt.legend()
plt.grid(True)
plt.show()
```

1.9 Additional Visual

[ ]:
```
accuracy_values = test_accuracies
error_values = [1 - accuracy for accuracy in accuracy_values]
print(error_values)


plt.figure(figsize=(10, 6))
plt.plot(k_values, error_values, label='Error Rate', marker='o', color='red')
plt.title('Error Rate for Different k Values in KNN: Health Insurance')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Error Rate')
plt.xticks(k_values)
```

```
plt.grid()
plt.legend()
plt.show()
```

[ ]: