Socioeconomic Influences

March 28, 2024

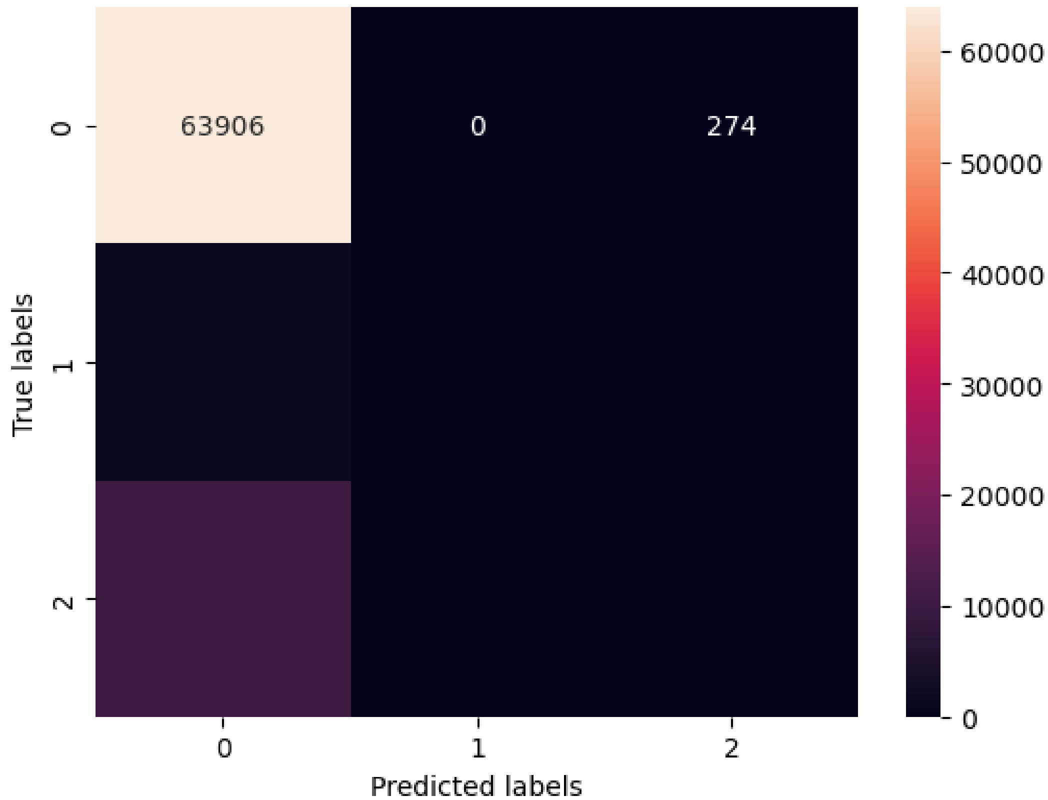
2. Socioeconomic Influences: In what ways can support vector machines (SVMs) elucidate the impact of socioeconomic and environmental factors on diabetes progression?

[2]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.feature_selection import SelectFromModel
import seaborn as sns  # Import seaborn for plotting
import matplotlib.pyplot as plt

# Assuming the 'diabetes_012_health_indicators_BRFSS2015.csv' file path is␣
 ↪correct and accessible
# Load the dataset
df = pd.read_csv('/Users/winnietaiwo/Desktop/
 ↪diabetes_012_health_indicators_BRFSS2015.csv')

# Corrected list of socioeconomic features
socioeconomic_features = ['Education', 'Income', 'BMI', 'Smoker',␣
 ↪'PhysActivity', 'Fruits', 'Veggies']

# Define features and target
X = df[socioeconomic_features]
y = df['Diabetes_012']  # Target variable

# Preprocess the data: scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3,␣
 ↪random_state=42)

# Increase max_iter and set dual=False to avoid future warnings
linear_svc = LinearSVC(C=0.01, penalty="l1", dual=False, max_iter=5000).
 ↪fit(X_train, y_train)
model = SelectFromModel(linear_svc, prefit=True)
```

```python
X_train_selected = model.transform(X_train)
X_test_selected = model.transform(X_test)

# Tuning using GridSearchCV
parameters = {'C': [0.01, 0.1, 1], 'max_iter': [5000]}  # Increased max_iter␣
↪for convergence
grid_search = GridSearchCV(LinearSVC(dual=False), parameters, cv=3, n_jobs=-1) ␣
↪# Reduced cv and use all cores with n_jobs=-1
grid_search.fit(X_train_selected, y_train)

# Best model after tuning
best_svm = grid_search.best_estimator_

# Predictions
predictions = best_svm.predict(X_test_selected)

# Evaluation, set zero_division parameter to avoid warnings
print(classification_report(y_test, predictions, zero_division=0))
print("Confusion Matrix:")
conf_matrix = confusion_matrix(y_test, predictions)
print(conf_matrix)

# Confusion matrix visualization
sns.heatmap(conf_matrix, annot=True, fmt='g')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()

# Features selected
selected_features = X.columns[model.get_support()]
print("Number of features selected: %d" % len(selected_features))
print('Selected features:', selected_features)

# There seems to be some code below that was perhaps pasted by accident.
# It should be removed if it is not part of the intended script.
```

```
              precision    recall  f1-score   support

         0.0       0.84      1.00      0.91     64180
         1.0       0.00      0.00      0.00      1425
         2.0       0.30      0.01      0.02     10499

    accuracy                           0.84     76104
   macro avg       0.38      0.34      0.31     76104
weighted avg       0.75      0.84      0.77     76104

Confusion Matrix:
```

```
[[63906     0   274]
 [ 1416     0     9]
 [10376     0   123]]
```




```
Number of features selected: 7
Selected features: Index(['Education', 'Income', 'BMI', 'Smoker',
'PhysActivity', 'Fruits',
       'Veggies'],
      dtype='object')
```

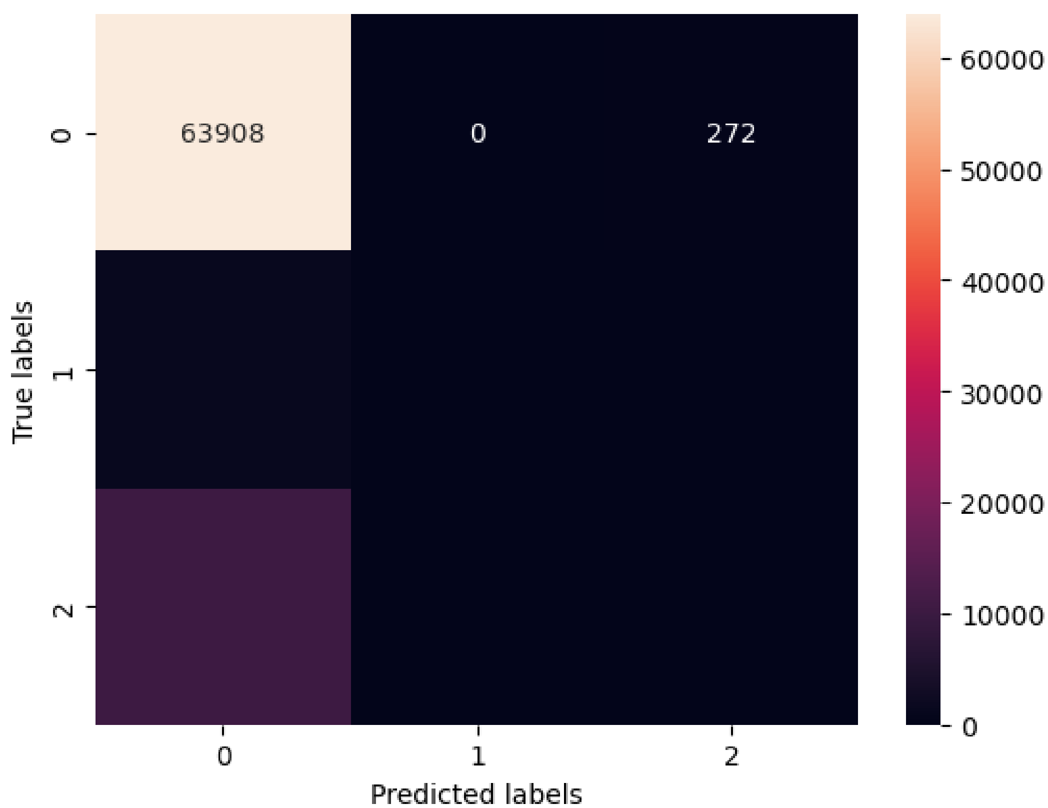
[4]:

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.feature_selection import SelectFromModel
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('/Users/winnietaiwo/Desktop/
 ↪diabetes_012_health_indicators_BRFSS2015.csv')
```

```python
# Define socioeconomic features and target
socioeconomic_features = ['Education', 'Income', 'BMI', 'Smoker',␣
↪'PhysActivity', 'Fruits', 'Veggies']
X = df[socioeconomic_features]
y = df['Diabetes_012']

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3,␣
↪random_state=42)

# Initialize the LinearSVC model with L1 penalty
linear_svc = LinearSVC(C=0.01, penalty="l1", dual=False, max_iter=5000)

# Feature selection using SelectFromModel
model = SelectFromModel(linear_svc, prefit=False)

# Setup GridSearchCV for hyperparameter tuning
parameters = {'C': [0.01, 0.1, 1], 'max_iter': [5000]}
grid_search = GridSearchCV(LinearSVC(penalty="l1", dual=False), parameters,␣
↪cv=3, n_jobs=-1)
grid_search.fit(X_train, y_train)

# Get the best model after hyperparameter tuning
best_svm = grid_search.best_estimator_

# Apply feature selection using the best model
model = SelectFromModel(best_svm, prefit=True)
X_train_selected = model.transform(X_train)
X_test_selected = model.transform(X_test)

# Make predictions with the tuned model
predictions = best_svm.predict(X_test_selected)

# Evaluate the model
print(classification_report(y_test, predictions, zero_division=0))
print("Confusion Matrix:")
conf_matrix = confusion_matrix(y_test, predictions)

# Visualize the confusion matrix
sns.heatmap(conf_matrix, annot=True, fmt='g')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
```

```
plt.show()

# Analyze the coefficients of the selected features for interpretation
selected_features = X.columns[model.get_support()]
coefficients = best_svm.coef_
coefficients_dict = {feature: coef for feature, coef in zip(selected_features,␣
 ↪coefficients.flatten())}
print("Selected features and their coefficients:")
print(coefficients_dict)
```

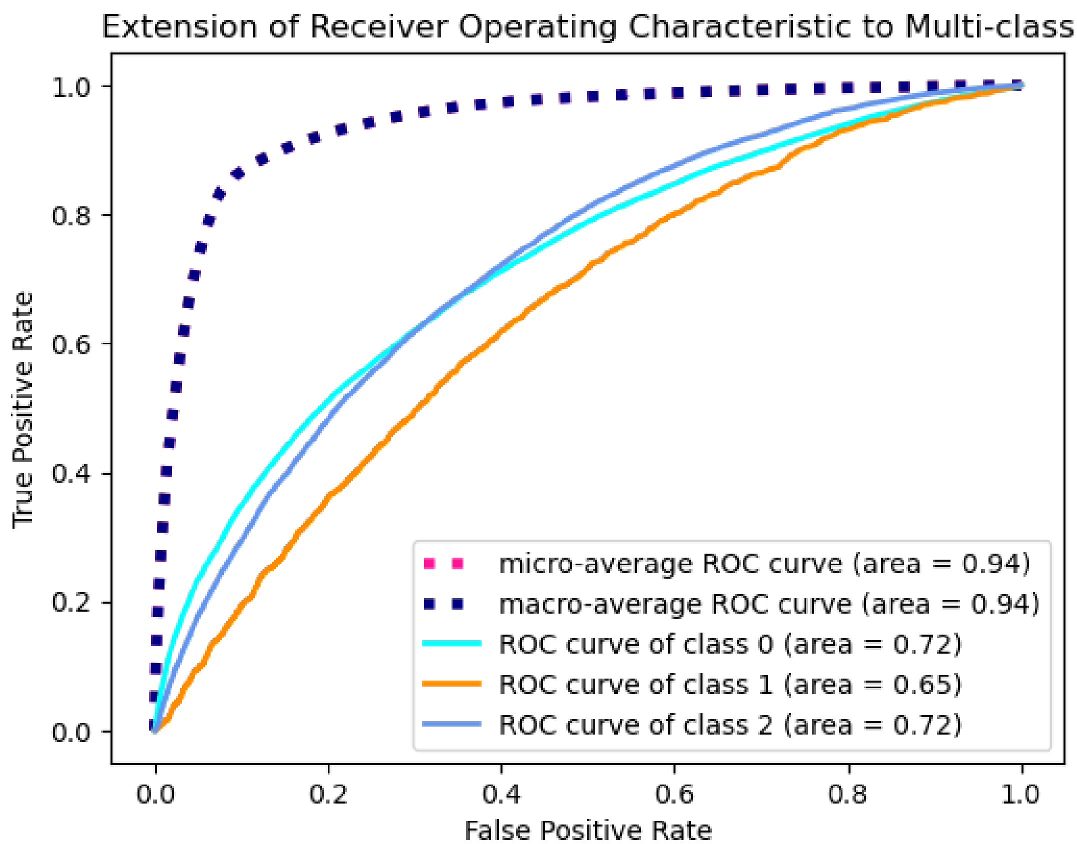
```
              precision    recall  f1-score   support

         0.0       0.84      1.00      0.91     64180
         1.0       0.00      0.00      0.00      1425
         2.0       0.30      0.01      0.02     10499

    accuracy                           0.84     76104
   macro avg       0.38      0.34      0.31     76104
weighted avg       0.75      0.84      0.77     76104

Confusion Matrix:
```



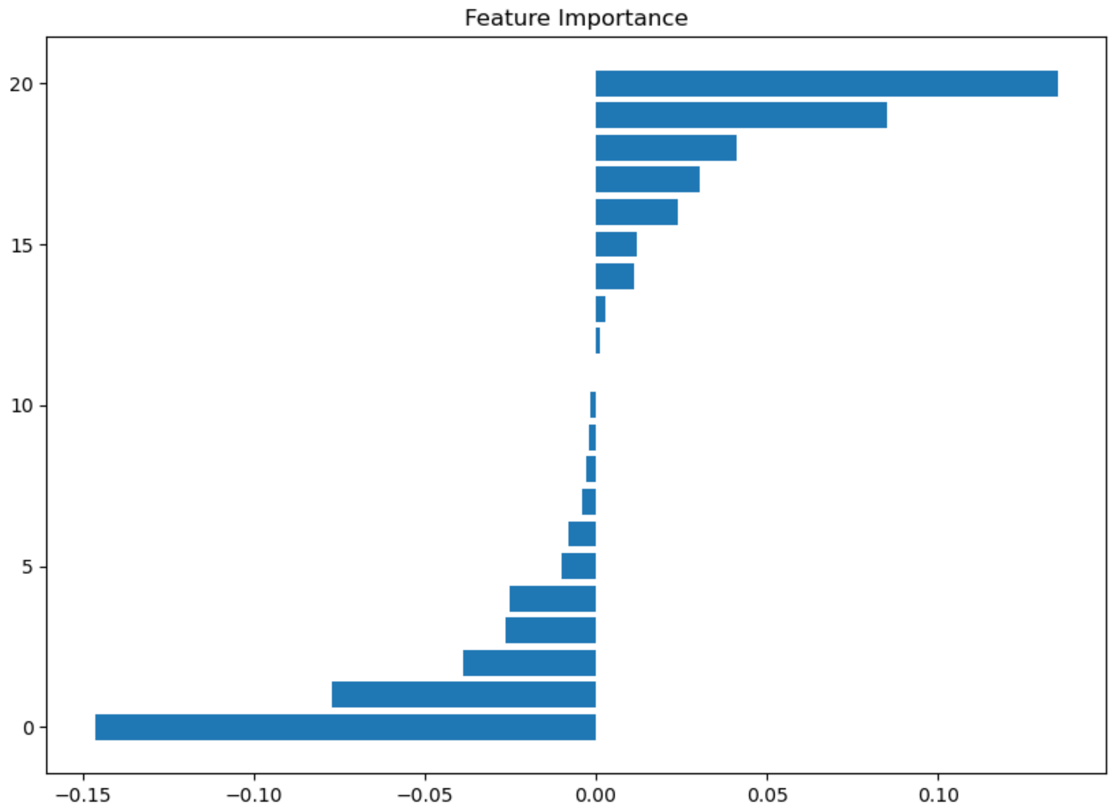
```
Selected features and their coefficients:
{'Education': 0.030356225770398745, 'Income': 0.08523965605644407, 'BMI':
-0.1464667801401856, 'Smoker': -0.025281873322904963, 'PhysActivity':
0.04108137880625749, 'Fruits': -0.003018693102231555, 'Veggies':
0.011983520941558272}
```

[8]:
```python
from itertools import cycle
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt
import numpy as np

# Binarize the output
y_test_bin = label_binarize(y_test, classes=np.unique(y_test))
n_classes = y_test_bin.shape[1]

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Aggregate all false positive rates
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

# Then interpolate all ROC curves at these points
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])

# Finally average it and compute AUC
mean_tpr /= n_classes

fpr["macro"], tpr["macro"], _ = roc_curve(y_test_bin.ravel(), y_score.ravel())
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot all ROC curves
plt.figure()
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["micro"]),
```

```
         color='deeppink', linestyle=':', linewidth=4)

plt.plot(fpr["macro"], tpr["macro"],
         label='macro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["macro"]),
         color='navy', linestyle=':', linewidth=4)

colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i, roc_auc[i]))

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Extension of Receiver Operating Characteristic to Multi-class')
plt.legend(loc="lower right")
plt.show()

# Assuming 'best_svm' is your trained SVM model and 'socioeconomic_features'␣
 ↪are your feature names
feature_importance = best_svm.coef_.ravel()

# Now we can plot the feature importances:
sorted_idx = np.argsort(feature_importance)
plt.figure(figsize=(10, 7))
plt.title('Feature Importance')
plt.barh(range(len(sorted_idx)), feature_importance[sorted_idx], align='center')
plt.yticks(range(len(sorted_idx)), np.array(socioeconomic_features)[sorted_idx])
plt.xlabel('Coefficient Value')
plt.show()
```




```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[8], line 69
     67 plt.title('Feature Importance')
     68 plt.barh(range(len(sorted_idx)), feature_importance[sorted_idx],␣
 ↪align='center')
---> 69 plt.yticks(range(len(sorted_idx)), np.
 ↪array(socioeconomic_features)[sorted_idx])
     70 plt.xlabel('Coefficient Value')
     71 plt.show()

IndexError: index 15 is out of bounds for axis 0 with size 7
```


Feature Importance
20
15
10
5
0
−0.15
−0.10
−0.05
0.00
0.05
0.10

[ ]: