week5_checkin

November 1, 2024

1 Week 5 Check-In

1.1 Team Spotiflies: Joanna, Aaron, Aubrey, Kennedy, Aster, Ethan

GitHub Link: https://github.com/ketexon/csm148-spotiflies

[1]:
```
%pip install pandas numpy matplotlib seaborn scikit-learn mlxtend plotly␣
↪nbformat
```

```
Requirement already satisfied: pandas in ./.venv/lib/python3.11/site-packages
(2.2.3)
Requirement already satisfied: numpy in ./.venv/lib/python3.11/site-packages
(2.1.2)
Requirement already satisfied: matplotlib in ./.venv/lib/python3.11/site-
packages (3.9.2)
Requirement already satisfied: seaborn in ./.venv/lib/python3.11/site-packages
(0.13.2)
Requirement already satisfied: scikit-learn in ./.venv/lib/python3.11/site-
packages (1.5.2)
Requirement already satisfied: mlxtend in ./.venv/lib/python3.11/site-packages
(0.23.1)
Requirement already satisfied: plotly in ./.venv/lib/python3.11/site-packages
(5.24.1)
Requirement already satisfied: nbformat in ./.venv/lib/python3.11/site-packages
(5.10.4)
Requirement already satisfied: python-dateutil>=2.8.2 in
./.venv/lib/python3.11/site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in ./.venv/lib/python3.11/site-
packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in ./.venv/lib/python3.11/site-
packages (from pandas) (2024.2)
Requirement already satisfied: contourpy>=1.0.1 in ./.venv/lib/python3.11/site-
packages (from matplotlib) (1.3.0)
Requirement already satisfied: cycler>=0.10 in ./.venv/lib/python3.11/site-
packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in ./.venv/lib/python3.11/site-
packages (from matplotlib) (4.54.1)
Requirement already satisfied: kiwisolver>=1.3.1 in ./.venv/lib/python3.11/site-
packages (from matplotlib) (1.4.7)
```

```
Requirement already satisfied: packaging>=20.0 in ./.venv/lib/python3.11/site-packages (from matplotlib) (24.1)
Requirement already satisfied: pillow>=8 in ./.venv/lib/python3.11/site-packages (from matplotlib) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in ./.venv/lib/python3.11/site-packages (from matplotlib) (3.2.0)
Requirement already satisfied: scipy>=1.6.0 in ./.venv/lib/python3.11/site-packages (from scikit-learn) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in ./.venv/lib/python3.11/site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in ./.venv/lib/python3.11/site-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: tenacity>=6.2.0 in ./.venv/lib/python3.11/site-packages (from plotly) (9.0.0)
Requirement already satisfied: fastjsonschema>=2.15 in ./.venv/lib/python3.11/site-packages (from nbformat) (2.20.0)
Requirement already satisfied: jsonschema>=2.6 in ./.venv/lib/python3.11/site-packages (from nbformat) (4.23.0)
Requirement already satisfied: jupyter-core!=5.0.*,>=4.12 in ./.venv/lib/python3.11/site-packages (from nbformat) (5.7.2)
Requirement already satisfied: traitlets>=5.1 in ./.venv/lib/python3.11/site-packages (from nbformat) (5.14.3)
Requirement already satisfied: attrs>=22.2.0 in ./.venv/lib/python3.11/site-packages (from jsonschema>=2.6->nbformat) (24.2.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in ./.venv/lib/python3.11/site-packages (from jsonschema>=2.6->nbformat) (2024.10.1)
Requirement already satisfied: referencing>=0.28.4 in ./.venv/lib/python3.11/site-packages (from jsonschema>=2.6->nbformat) (0.35.1)
Requirement already satisfied: rpds-py>=0.7.1 in ./.venv/lib/python3.11/site-packages (from jsonschema>=2.6->nbformat) (0.20.1)
Requirement already satisfied: platformdirs>=2.5 in ./.venv/lib/python3.11/site-packages (from jupyter-core!=5.0.*,>=4.12->nbformat) (4.3.6)
Requirement already satisfied: six>=1.5 in ./.venv/lib/python3.11/site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)

[notice] A new release of pip available: 22.3 -> 24.3.1
[notice] To update, run: pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```python
# SETUP data set like in week 4:


import pandas as pd
import numpy as np
```

```python
import math
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
import plotly.io as pio

pio.renderers.default = "pdf"

# Reading in the cleaned data from previous week check in
spotify = pd.read_csv("csv_outputs/cleaned_spotify.csv")

# select the variables of interest
selected_spotify = spotify[['mode', 'valence', 'tempo']]
selected_spotify

random_seed = 42
response = 'mode'
predictor = 'valence'

# Splitting the data
# First split: separate out 20% for the test set
spotify_train_val, spotify_test = train_test_split(selected_spotify,␣
 ↪test_size=0.2, random_state=random_seed)

# Second split: separate remaining 80% into 60% training and 40% validation
spotify_train, spotify_val = train_test_split(spotify_train_val, test_size=0.
 ↪25, random_state=random_seed)  # 0.25 * 0.8 = 0.2

# Reshape the data to fit the model
X_train = spotify_train.drop(columns=[response, "tempo"])
y_train = spotify_train[response]

# fit the model and list intercept and coefficient
logistic_reg = LogisticRegression(solver='liblinear')
logistic_reg.fit(X=X_train,y=y_train)

# generate values for plotting the curve as a DataFrame with the same column␣
 ↪name
x_values = pd.DataFrame(np.linspace(0, 1, 100), columns=[predictor])  # Use␣
 ↪'valence' as the column name

# Now you can predict the probabilities without the feature name issue
y_values = logistic_reg.predict_proba(x_values)[:, 1]
```

1.1.1 KNN Algorithm:

[3]:
```
# Import necessary libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# Define the response and predictor variables
response = 'mode'
predictors = ['valence', 'tempo']  # Use both 'valence' and 'tempo' as␣
↪predictors

# Use the same train-test split as before, selecting both predictor columns
X_train = spotify_train[predictors]
y_train = spotify_train[response]

X_val = spotify_val[predictors]
y_val = spotify_val[response]

# Initialize the KNN model
# Set n_neighbors to the desired number (e.g., 5) - you can tune this␣
↪hyperparameter later
knn = KNeighborsClassifier(n_neighbors=5)

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

# Predict the values on the validation set
y_val_pred = knn.predict(X_val)

# Evaluate the model
print("Validation Accuracy:", accuracy_score(y_val, y_val_pred))
print(classification_report(y_val, y_val_pred))
```

```
Validation Accuracy: 0.6484210526315789
              precision    recall  f1-score   support

           0       0.52      0.42      0.46      8276
           1       0.70      0.78      0.74     14524

    accuracy                           0.65     22800
   macro avg       0.61      0.60      0.60     22800
weighted avg       0.64      0.65      0.64     22800
```

Unfortunately our KNN model's accuracy `valence` and `mode` variables isn't very good, but we can continue to check the validity of our model with the confusion matrix and other metrics.

1.1.2 Calculating the Confusion Matrix + Metrics

[4]:
```
# Calculate the confusion matrix using the validation set with the new response␣
↪variable
y_pred = knn.predict(spotify_val[predictors])  # Use the KNN model to predict␣
↪based on 'valence' and 'tempo'
y_true = spotify_val[response]

# Calculate and print metrics
conf = metrics.confusion_matrix(y_true=y_true, y_pred=y_pred)
print('Confusion Matrix:\n', conf)
print('Prediction Accuracy:', metrics.accuracy_score(y_true=y_true,␣
↪y_pred=y_pred))
print('Prediction Error:', 1 - metrics.accuracy_score(y_true=y_true,␣
↪y_pred=y_pred))
print('True Positive Rate (Recall):', metrics.recall_score(y_true=y_true,␣
↪y_pred=y_pred))
print('True Negative Rate (Specificity):', metrics.recall_score(y_true=y_true,␣
↪y_pred=y_pred, pos_label=0))
print('F1 Score:', metrics.f1_score(y_true=y_true, y_pred=y_pred))
```

```
Confusion Matrix:
 [[ 3470  4806]
 [ 3210 11314]]
Prediction Accuracy: 0.6484210526315789
Prediction Error: 0.3515789473684211
True Positive Rate (Recall): 0.7789865050950151
True Negative Rate (Specificity): 0.4192846785886902
F1 Score: 0.7384153504764391
```

Our model seems to be predicting all values as a positive based on the confusion matrix, which is a sign that the model doesn't fit our data very well. However, we can continue to investigate using the ROC Curve and AUC.

1.1.3 ROC Curve + AUC Calculation

[5]:
```
import plotly.express as px

# Make sure KNN was initialized with probability=True for this to work
# Create the ROC curve variables for KNN
knn_fpr_sample, knn_tpr_sample, knn_thresholds_sample = metrics.roc_curve(
    spotify_val[response], knn.predict_proba(spotify_val[predictors])[:, 1]
)

# Calculate AUC for KNN
knn_auc_sample = metrics.roc_auc_score(
    spotify_val[response], knn.predict_proba(spotify_val[predictors])[:, 1]
)
```

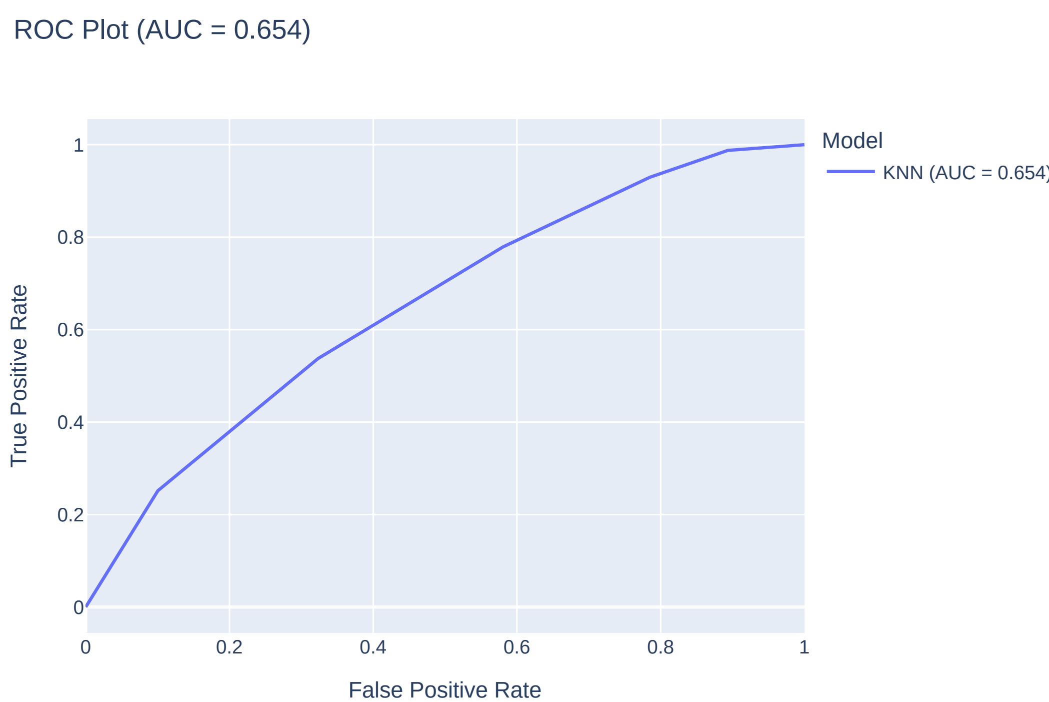
```
print('KNN AUC:', knn_auc_sample.round(3))

# Prepare DataFrame for plotting
roc_knn_sample = pd.DataFrame({
    'False Positive Rate': knn_fpr_sample,
    'True Positive Rate': knn_tpr_sample,
    'Model': f'KNN (AUC = {knn_auc_sample:.3f})'
}, index=knn_thresholds_sample)

# Plot the ROC curve with AUC in the title
fig = px.line(
    roc_knn_sample,
    y='True Positive Rate',
    x='False Positive Rate',
    color='Model',
    width=700,
    height=500,
    title=f"ROC Plot (AUC = {knn_auc_sample:.3f})"
)

# Show plot
fig.show()
```

```
KNN AUC: 0.654
```



Loading [MathJax]/extensions/MathMenu.js

Based on the AUC, we have a sensitivity rating of about 0.654, which is basically equivalent to making random guesses, so our model is probably not a good fit at all for the relationship between the `mode` and `valence` variables. The logistic model is probably not a good predictor model for our data.

1.1.4 5-Fold CV + AUC Calculation

[6]:
```python
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone

# Initialize Stratified K-Folds
skfolds = StratifiedKFold(n_splits=5)
i = 1
X = spotify_val[predictors]
y = spotify_val[response]

# Perform Stratified K-Fold Cross-Validation
for train_index, test_index in skfolds.split(X, y):
    # Clone the KNN model for each fold
    clone_knn = clone(knn)

    # Split data into training and test sets for this fold
    X_train_folds = X.iloc[train_index]
    y_train_folds = y.iloc[train_index]
    X_test_fold = X.iloc[test_index]
    y_test_fold = y.iloc[test_index]

    # Train the cloned model
    clone_knn.fit(X_train_folds, y_train_folds)

    # Get predictions and calculate probabilities for AUC
    y_pred = clone_knn.predict(X_test_fold)
    y_pred_proba = clone_knn.predict_proba(X_test_fold)[:, 1]

    # Calculate AUC and Accuracy for this fold
    auc_sample = metrics.roc_auc_score(y_test_fold, y_pred_proba)
    accuracy = metrics.accuracy_score(y_test_fold, y_pred)

    # Display results
    print(f'Fold: {i}')
    print('AUC:', auc_sample)
    print('Accuracy:', accuracy)

    i += 1
```

```
Fold: 1
```

```
AUC: 0.5912232789596019
Accuracy: 0.6232456140350877
Fold: 2
AUC: 0.5833472864266733
Accuracy: 0.6083333333333333
Fold: 3
AUC: 0.5915410974931232
Accuracy: 0.6160087719298246
Fold: 4
AUC: 0.5968416783231328
Accuracy: 0.6190789473684211
Fold: 5
AUC: 0.5833654604343832
Accuracy: 0.6171052631578947
```

We ended up picking the default threshold of 0.5 as a starting point for a model. With it, we were able to get an accuracy of ~0.65 and an AUC ~0.654, which means that the logistic model does only slightly better than random guessing at predicting our data. We found that the accuracy is the same as the class imbalance for the `mode` variable, so the small boost above 0.5 is likely due to that, and not actually our model performing well. In conclusion, the relationship between `valence` and `mode` is not well modelled by the logistic predictor model.