03_baseline_model_tidy

March 9, 2024

1 Baseline Model

2 Import Libraries

[10]:
```
import sys

# Data Manipulation
import numpy as np
import pandas as pd

# Machine Learning Model
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.metrics import roc_auc_score
from sklearn.compose import ColumnTransformer

# Class Imbalance
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as ImbPipeline

# Model Saving
import joblib
```

3 Load Data

Data is loaded from CSV files into Pandas data frames for ease of manipulation and analysis. Indexing by `respondent_id` helps keep track of each individual's data across multiple datasets.

[11]:
```
# Load training and test datasets
training_set_features = pd.read_csv('../data/clean/training_set_features.csv',␣
 ↪index_col="respondent_id")
training_set_labels = pd.read_csv('../data/clean/training_set_labels.csv',␣
 ↪index_col="respondent_id")
test_set_features = pd.read_csv('../data/clean/test_set_features.csv',␣
 ↪index_col="respondent_id")
```

4 Prepare Data

Separates the features from the target variables for both `h1n1_vaccine` and `seasonal_vaccine` predictions.

This step is essential for supervised learning, where the model needs to learn the relationship between the features and the target.

[12]:
```
# Separate features and targets for training
features = training_set_features
h1n1_target = training_set_labels['h1n1_vaccine']
seasonal_target = training_set_labels['seasonal_vaccine']
categorical_features = features.columns  # Assuming all features are categorical
```

5 Preprocessing Pipeline

Sets up a preprocessing pipeline with one-hot encoding for categorical features to convert categorical variables into a form that could be provided to Machine Learning algorithms to do a better job in prediction. This standardizes the data preparation process and ensures consistency in data transformation.

[13]:
```
# Define preprocessing for categorical data
categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Combine preprocessing steps
preprocessor = ColumnTransformer(transformers=[
    ('cat', categorical_transformer, categorical_features)
])
```

6 Model Pipelines

Constructs separate pipelines for `h1n1_vaccine` and `seasonal_vaccine` predictions, integrating preprocessing, handling class imbalance (for `h1n1_vaccine`), and logistic regression classification.

Pipelines streamline the process from raw data to predictions, ensuring reproducibility and ease of model updates.

[14]:
```
# Pipeline for h1n1 vaccine prediction
pipeline_h1n1 = ImbPipeline(steps=[
    ('preprocessor', preprocessor),
    ('oversample', SMOTE(sampling_strategy='auto', random_state=42)),
    ('classifier', LogisticRegression(random_state=42, max_iter=1000))
])

# Pipeline for seasonal vaccine prediction
pipeline_seasonal = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(random_state=42, max_iter=1000))
])
```

7 Split Dataset

Divides the dataset into `training` and `validation` sets to train the model on one set of data and validate its performance on a separate set. This helps in assessing the model's ability to generalize to unseen data (e.g. `test` set or unseen datasets)

[15]:

```
# Split data into training and validation sets
X_train, X_val, y_train_h1n1, y_val_h1n1, y_train_seasonal, y_val_seasonal =␣
 ↪train_test_split(
    features, h1n1_target, seasonal_target, test_size=0.2, random_state=42)
```

8 Fit/Train Models

Trains the logistic regression models within their respective pipelines on the training data. This step is where the model learns the relationship between the features and the target variable.

[16]:

```
# Train models
pipeline_h1n1.fit(X_train, y_train_h1n1)
pipeline_seasonal.fit(X_train, y_train_seasonal)
```

[16]:

```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('cat',
                                                  Pipeline(steps=[('onehot',
OneHotEncoder(handle_unknown='ignore'))]),
                                                  Index(['Unamed: 0',
'h1n1_concern', 'h1n1_knowledge',
       'behavioral_antiviral_meds', 'behavioral_avoidance',
       'behavioral_face_mask', 'behavioral_wash_hands',
       'behavioral_large_gatherings', 'behavioral_outside_home',
       'behav…
       'opinion_seas_risk', 'opinion_seas_sick_from_vacc', 'age_group',
       'education', 'race', 'sex', 'income_poverty', 'marital_status',
       'rent_or_own', 'employment_status', 'hhs_geo_region', 'census_msa',
       'household_adults', 'household_children', 'employment_industry',
       'employment_occupation'],
      dtype='object'))])),
                ('classifier',
                 LogisticRegression(max_iter=1000, random_state=42))])
```

9 Make Predictions on the Validation Set and Evaluate Model

Make predictions on the validation set and evaluate the model's performance using the `ROC-AUC` score. Uses the `ROC-AUC` metric to evaluate model performance on the validation set. `ROC-AUC` is chosen for its ability to handle imbalanced classes and to provide a measure of how well the model distinguishes between classes. It's also crucial for understanding how well the model might perform on the test set and future unseen data.

[17]:

```python
# Validate models using ROC-AUC on the validation set
probabilities_h1n1_val = pipeline_h1n1.predict_proba(X_val)[:, 1]  # Probabilities for h1n1_vaccine
probabilities_seasonal_val = pipeline_seasonal.predict_proba(X_val)[:, 1]  # Probabilities for seasonal_vaccine
roc_auc_h1n1 = roc_auc_score(y_val_h1n1, probabilities_h1n1_val)
roc_auc_seasonal = roc_auc_score(y_val_seasonal, probabilities_seasonal_val)
print(f"ROC-AUC for H1N1 Vaccine Prediction on Validation Set: {roc_auc_h1n1}")
print(f"ROC-AUC for Seasonal Vaccine Prediction on Validation Set: {roc_auc_seasonal}")
```

```
ROC-AUC for H1N1 Vaccine Prediction on Validation Set: 0.8206701880005715
ROC-AUC for Seasonal Vaccine Prediction on Validation Set: 0.8556704842798477
```

10 Make Predictions on Test Set

Applies the trained models to the test set to generate predictions. This step is crucial for the competition and assessing how the models might perform in real-world scenarios or on unseen data.

[18]:

```python
# Making predictions on the test set for both h1n1 and seasonal vaccines
probabilities_h1n1_test = pipeline_h1n1.predict_proba(test_set_features)[:, 1] # Probabilities for h1n1_vaccine
probabilities_seasonal_test = pipeline_seasonal.predict_proba(test_set_features)[:, 1]  # Probabilities for seasonal_vaccine
```

11 Submission

Saves a dated submission `.csv` file suitable for uploading to the DrivenData competition website.

[19]:

```python
# Prepare submission DataFrame with current date
from datetime import datetime
current_date = datetime.now().strftime("%Y-%m-%d")
submission_df = pd.DataFrame({
    "respondent_id": test_set_features.index,
    "h1n1_vaccine": probabilities_h1n1_test,
    "seasonal_vaccine": probabilities_seasonal_test
})
submission_file_path = f"../submissions/submission_{current_date}.csv"
```

```
submission_df.to_csv(submission_file_path, index=False)
print(f"Submission file with {submission_df.shape[0]} rows saved to␣
  ↪{submission_file_path}")
```

```
Submission file with 26708 rows saved to
../submissions/submission_2024-03-09.csv
```

12 Save Models

Saves the models for loading in future analyses.

[20]:

```
# Save models to files
model_path_h1n1 = "../models/pipeline_h1n1.joblib"
model_path_seasonal = "../models/pipeline_seasonal.joblib"
joblib.dump(pipeline_h1n1, model_path_h1n1)
joblib.dump(pipeline_seasonal, model_path_seasonal)
print(f"Model saved to {model_path_h1n1}")
print(f"Model saved to {model_path_seasonal}")
```

```
Model saved to ../models/pipeline_h1n1.joblib
Model saved to ../models/pipeline_seasonal.joblib
```