Assigment 8: Choose Your ML Problem and Data

In this unit's lab, you will implement a model to solve a machine learning problem of your choosing. First, you will have to make some decisions, such as which model to choose and which data preparation techniques may be necessary, and formulate a project plan accordingly.

In this assignment, you will select a data set and choose a predictive problem that the data set supports. You will then inspect the data with your problem in mind and begin to formulate your project plan. You will create this project plan in the written assignment that follows.

Import Packages

Before you get started, import a few packages. You can import additional packages that you have used in this course that you may need for this task.

In [7]:
```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import plot_roc_curve, accuracy_score, roc_auc_score
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
```

Step 1: Choose Your Data Set and Load the Data

You will have the option to choose one of four data sets that you have worked with in this program:

- The "adult" data set that contains Census information from 1994: `adultData.csv`
- Airbnb NYC "listings" data set: `airbnbListingsData.csv`
- World Happiness Report (WHR) data set: `WHR2018Chapter2OnlineData.csv`
- Book Review data set: `bookReviewsData.csv`

Note that these are variations of the data sets that you have worked with in this program. For example, some do not include some of the preprocessing necessary for specific models.

Load the Data Set

The code cell below contains filenames (path + filename) for each of the four data sets available to you.

Task: In the code cell below, use the same method you have been using to load the data using `pd.read_csv()` and save it to DataFrame `df`.

You can load each file as a new DataFrame to inspect the data before choosing your data set.

In [8]:
```
# File names of the four data sets
adultDataSet_filename = os.path.join(os.getcwd(), "data", "adultData.csv")
airbnbDataSet_filename = os.path.join(os.getcwd(), "data", "airbnbListingsData.csv")
WHRDataSet_filename = os.path.join(os.getcwd(), "data", "WHR2018Chapter2OnlineData.csv")
bookReviewDataSet_filename = os.path.join(os.getcwd(), "data", "bookReviewsData.csv")

df = pd.read_csv(bookReviewDataSet_filename, header=0)

df.head()
```

Out[8]:

	Review	Positive Review
0	This was perhaps the best of Johannes Steinhof...	True
1	This very fascinating book is a story written ...	True
2	The four tales in this collection are beautifu...	True
3	The book contained more profanity than I expec...	False
4	We have now entered a second time of deep conc...	True

Step 2: Choose Your Predictive Problem and Label

Now that you have chosen your data set, you can:

1. Choose what you would like to predict (i.e. the label)
2. Identify your problem type: is it a classification or regression problem?

Task: In the markdown cell below, state what you are predicting (the label) and whether this is a classification or regression problem.

Answer:

1. I would like to predict the reviews given and if those reviews are either positive or negative.
2. This is a classification problem since it requires inputs being placed into two or more classes. This means its also a binary one since it is either true or false (positive or negative).

Step 3: Inspect Your Data

In the code cell below, use some of the techniques you have learned in this course to take a look at your data. As you are investigating your data, consider the following to help you formulate your project plan:

1. What are my features?
2. Which model (or models) should I select that is appropriate for my machine learning problem and data?
3. Which data preparation techniques may be needed for my model (e.g. perform one-hot encoding)?
4. Which techniques should I use to evaluate my model's performance and improve my model?

Note: You will use this notebook to take a glimpse at your data to help you start making some considerations. In the written assignment you will outline your project plan, and in the lab assignment you will perform a deeper exploratory analysis of the data before implementing data preparation and feature engineering techniques.

Task: Use the techniques you have learned in this course to inspect your data.

Note: You can add code cells if needed by going to the **Insert** menu and clicking on **Insert Cell Below** in the drop-drown menu.

Creating Labeled Examples

In [9]:
```
# creating labeled examples from df
y = df['Positive Review']
X = df['Review']
```

In [10]:
```
X.head()
```

Out[10]:
```
0    This was perhaps the best of Johannes Steinhof...
1    This very fascinating book is a story written ...
2    The four tales in this collection are beautifu...
3    The book contained more profanity than I expec...
4    We have now entered a second time of deep conc...
Name: Review, dtype: object
```

In [11]:
```
X.shape
```

Out[11]:
```
(1973,)
```

Split Labeled Examples into Training and Test Sets

In [12]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.20
,random_state=1234)
X_train.shape
```

Out[12]:
```
(1578,)
```

Implement TF-IDF Vectorizer to Transform Text

In [13]:
```
# 1. Create a TfidfVectorizer object and save it to the variable 'tfidf_vectorizer'
tfidf_vectorizer = TfidfVectorizer()

# 2. Fit the vectorizer to X_train
tfidf_vectorizer.fit(X_train)

# 3. Using the fitted vectorizer, transform the training data and save the data to variable 'X_train_tfidf'
X_train_tfidf = tfidf_vectorizer.transform(X_train)

# 4. Using the fitted vectorizer, transform the test data and save the data to variable 'X_test_tfidf'
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

In [14]:
```
print(X_test_tfidf)
```

```
  (0, 18965)    0.059491023406618646
  (0, 18727)    0.08752131471965732
  (0, 18642)    0.03533743581074492
  (0, 18593)    0.03466402255636781
  (0, 18539)    0.10400005525124341
  (0, 18496)    0.09274785194457173
  (0, 18455)    0.0276659188493222
  (0, 18126)    0.051656463721148134
  (0, 17733)    0.1213832032593689
  (0, 17680)    0.1213832032593689
  (0, 17618)    0.06952952155086067
  (0, 17302)    0.047594430972735094
  (0, 17259)    0.0725088662135299
  (0, 17226)    0.04030762713391491
  (0, 17133)    0.1295180018700078
  (0, 17117)    0.049162328805929446
  (0, 17104)    0.07672093808666652
  (0, 17066)    0.047152835152653096
  (0, 17061)    0.04603216704495873
  (0, 17053)    0.043249080127104864
  (0, 17044)    0.03390923842467748
  (0, 17040)    0.02324158038379852
  (0, 16805)    0.05557793152499216
  (0, 16288)    0.1104156579703061
  (0, 16266)    0.05275501870340633
  :     :
  (394, 16266)  0.11591490667341263
  (394, 16183)  0.16526647998794597
  (394, 14669)  0.21441548095429677
  (394, 11847)  0.041179983410956134
  (394, 11711)  0.11266334411684766
  (394, 10947)  0.45702407068616013
  (394, 10044)  0.21075247713547102
  (394, 9429)   0.15769702241845307
  (394, 8834)   0.18665425823869614
  (394, 8715)   0.046490734449862806
  (394, 8146)   0.12026637617557144
  (394, 8045)   0.16211290791224467
  (394, 7953)   0.12465219981184637
  (394, 7346)   0.21441548095429677
  (394, 7117)   0.1884375404488169
  (394, 6962)   0.050473977587676314
  (394, 4766)   0.2667068086286382
  (394, 3090)   0.18031559754953025
  (394, 2641)   0.16924411383550905
  (394, 2587)   0.05758922117953014
  (394, 1914)   0.39331184549337556
  (394, 1898)   0.11328978234234187
  (394, 1344)   0.060280163335188666
  (394, 1248)   0.23485069603224593
  (394, 1240)   0.06240793416993048
```

Fit a Logistic Regression Model to the Transformed Training Data and Evaluate the Model

In [15]:
```
# 1. Create the LogisticRegression model object
model = LogisticRegression( max_iter=200)

# 2. Fit the model to the transformed training data
model.fit(X_train_tfidf, y_train)

# 3. Use the predict_proba() method to make predictions on the test data
probability_predictions = model.predict_proba(X_test_tfidf)[:,1]

# 4. Compute the area under the ROC curve for the test data.
auc = roc_auc_score(y_test, probability_predictions)
print('AUC on the test data: {:.4f}'.format(auc))

# 5. Compute the size of the resulting feature space
len_feature_space = len(tfidf_vectorizer.vocabulary_)
print('The size of the feature space: {0}'.format(len_feature_space))
```

```
AUC on the test data: 0.9161
The size of the feature space: 19029
```

Experiment with Different Document Frequency Values and Analyze the Results

In [18]:
```
for min_df in [2,34,25,14]:
    print('\nDocument Frequency Value: {0}'.format(min_df))


# 1. Create a TfidfVectorizer object and save it to the variable 'tfidf_vectorizer'
# Use the arguments: 'ngram_range=(1,2)'' and 'min_df=min_df'
tfidf_vectorizer = TfidfVectorizer(ngram_range=(1,2), min_df=min_df)

# 2. Fit the vectorizer to X_train
tfidf_vectorizer.fit(X_train)

# 3. Using the fitted vectorizer, transform the training data.
# Save the transformed training data to variable 'X_train_tfidf'
X_train_tfidf = tfidf_vectorizer.transform(X_train)

# 4. Using the fitted vectorizer, transform the test data.
# Save the transformed test data to variable 'X_test_tfidf'
X_test_tfidf = tfidf_vectorizer.transform(X_test)

# 5. Create the LogisticRegression model object and save it to variable 'model'.
# Call LogisticRegression() with the argument 'max_iter=200'
model = LogisticRegression(max_iter=200)

# 6. Fit the model to the transformed training data
model.fit(X_train_tfidf, y_train)

# 7. Use the predict_proba() method to make predictions on the transformed
# Save the second column to the variable 'probability_predictions'
probability_predictions = model.predict_proba(X_test_tfidf)[:,1]

# 8. Using roc_auc_score() function to compute the AUC.
# Save the result to the variable 'auc'
auc = roc_auc_score(y_test, probability_predictions)
print('AUC on the test data: {:.4f}'.format(auc))

# 9. Compute the size of the resulting feature space
# Save the result to the variable 'len_feature_space'
len_feature_space = len(tfidf_vectorizer.vocabulary_)
print('The size of the feature space: {0}'.format(len_feature_space))
```

```
Document Frequency Value: 2

Document Frequency Value: 34

Document Frequency Value: 25

Document Frequency Value: 14
AUC on the test data: 0.9176
The size of the feature space: 2901
```

Set up a TF-IDF + Logistic Regression Pipeline

In [19]:
```
from sklearn.pipeline import Pipeline
```

In [20]:
```
print('Begin ML pipeline...')
# 1. Define the list of steps:
s = [
("vectorizer", TfidfVectorizer(ngram_range=(1,2), min_df=10)),
("model", LogisticRegression(max_iter=200))
]

# 2. Define the pipeline:
model_pipeline = Pipeline(steps=s)
# We can use the pipeline the way would would use a model object
# when fitting the model on the training data testing on the test data:

# 3. Fit the pipeline to the training data
model_pipeline.fit(X_train, y_train)

# 4. Make predictions on the test data.
# Save the second column to the variable 'probability_predictions
probability_predictions = model_pipeline.predict_proba(X_test)[:,1]
print('End pipeline')
```

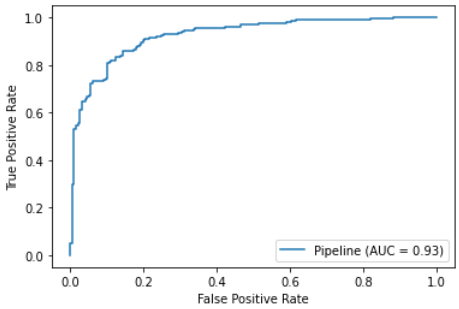
```
Begin ML pipeline...
End pipeline
```

In [21]:
```
# Evaluate the performance by computing the AUC
auc_score = roc_auc_score(y_test, probability_predictions)
print('AUC on the test data: {:.4f}'.format(auc_score))
```

```
AUC on the test data: 0.9254
```

In [22]:
```
from sklearn.metrics import plot_roc_curve
plot_roc_curve(model_pipeline, X_test, y_test)
```

Out[22]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f03dbf91ba8>
```



Perform a GridSearchCV on the Pipeline to Find the Best Hyperparameters

In [24]:
```
model_pipeline.get_params().keys()
```

Out[24]:
```
dict_keys(['memory', 'steps', 'verbose', 'vectorizer', 'model', 'vectorizer__analyzer', 'vectorizer__binary', 'vectorizer__decode_
error', 'vectorizer__dtype', 'vectorizer__encoding', 'vectorizer__input', 'vectorizer__lowercase', 'vectorizer__max_df', 'vectoriz
er__max_features', 'vectorizer__min_df', 'vectorizer__ngram_range', 'vectorizer__norm', 'vectorizer__preprocessor', 'vectorizer__s
mooth_idf', 'vectorizer__stop_words', 'vectorizer__strip_accents', 'vectorizer__sublinear_tf', 'vectorizer__token_pattern', 'vecto
rizer__tokenizer', 'vectorizer__use_idf', 'vectorizer__vocabulary', 'model__C', 'model__class_weight', 'model__dual', 'model__fit_
intercept', 'model__intercept_scaling', 'model__l1_ratio', 'model__max_iter', 'model__multi_class', 'model__n_jobs', 'model__penal
ty', 'model__random_state', 'model__solver', 'model__tol', 'model__verbose', 'model__warm_start'])
```

In [25]:
```
param_grid = {'vectorizer__ngram_range':[(1,1), (1,2)],
'model__C':[0.1, 1, 10]
}
param_grid
```

Out[25]:
```
{'model__C': [0.1, 1, 10], 'vectorizer__ngram_range': [(1, 1), (1, 2)]}
```

In [26]:
```
print('Running Grid Search...')

# 1. Run a Grid Search with 3-fold cross-validation and assign the output to the object 'grid_LR'.
grid = GridSearchCV(model_pipeline, param_grid=param_grid, cv=3, verbose=2,scoring = 'roc_auc')

# 2. Fit the model (grid_LR) on the training data and assign the fitted model to the variable 'grid_search_LR'
grid_search = grid.fit(X_train, y_train)

print('Done')
```

```
Running Grid Search...
Fitting 3 folds for each of 6 candidates, totalling 18 fits
[CV] model__C=0.1, vectorizer__ngram_range=(1, 1) ....................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] ..... model__C=0.1, vectorizer__ngram_range=(1, 1), total=   0.3s
[CV] model__C=0.1, vectorizer__ngram_range=(1, 1) ....................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.3s remaining:    0.0s

[CV] ..... model__C=0.1, vectorizer__ngram_range=(1, 1), total=   0.3s
[CV] model__C=0.1, vectorizer__ngram_range=(1, 1) ....................
[CV] ..... model__C=0.1, vectorizer__ngram_range=(1, 1), total=   0.3s
[CV] model__C=0.1, vectorizer__ngram_range=(1, 2) ....................
[CV] ..... model__C=0.1, vectorizer__ngram_range=(1, 2), total=   0.9s
[CV] model__C=0.1, vectorizer__ngram_range=(1, 2) ....................
[CV] ..... model__C=0.1, vectorizer__ngram_range=(1, 2), total=   0.9s
[CV] model__C=0.1, vectorizer__ngram_range=(1, 2) ....................
[CV] ..... model__C=0.1, vectorizer__ngram_range=(1, 2), total=   0.8s
[CV] model__C=1, vectorizer__ngram_range=(1, 1) ......................
[CV] ....... model__C=1, vectorizer__ngram_range=(1, 1), total=   0.3s
[CV] model__C=1, vectorizer__ngram_range=(1, 1) ......................
[CV] ....... model__C=1, vectorizer__ngram_range=(1, 1), total=   0.3s
[CV] model__C=1, vectorizer__ngram_range=(1, 1) ......................
[CV] ....... model__C=1, vectorizer__ngram_range=(1, 1), total=   0.3s
[CV] model__C=1, vectorizer__ngram_range=(1, 2) ......................
[CV] ....... model__C=1, vectorizer__ngram_range=(1, 2), total=   0.9s
[CV] model__C=1, vectorizer__ngram_range=(1, 2) ......................
[CV] ....... model__C=1, vectorizer__ngram_range=(1, 2), total=   0.9s
[CV] model__C=1, vectorizer__ngram_range=(1, 2) ......................
[CV] ....... model__C=1, vectorizer__ngram_range=(1, 2), total=   0.8s
[CV] model__C=10, vectorizer__ngram_range=(1, 1) .....................
[CV] ...... model__C=10, vectorizer__ngram_range=(1, 1), total=   0.3s
[CV] model__C=10, vectorizer__ngram_range=(1, 1) .....................
[CV] ...... model__C=10, vectorizer__ngram_range=(1, 1), total=   0.3s
[CV] model__C=10, vectorizer__ngram_range=(1, 1) .....................
[CV] ...... model__C=10, vectorizer__ngram_range=(1, 1), total=   0.3s
[CV] model__C=10, vectorizer__ngram_range=(1, 2) .....................
[CV] ...... model__C=10, vectorizer__ngram_range=(1, 2), total=   0.9s
[CV] model__C=10, vectorizer__ngram_range=(1, 2) .....................
[CV] ...... model__C=10, vectorizer__ngram_range=(1, 2), total=   0.9s
[CV] model__C=10, vectorizer__ngram_range=(1, 2) .....................
[CV] ...... model__C=10, vectorizer__ngram_range=(1, 2), total=   0.9s

[Parallel(n_jobs=1)]: Done  18 out of  18 | elapsed:   10.8s finished

Done
```

In [27]:
```
grid_search.best_estimator_
```

Out[27]:
```
Pipeline(memory=None,
         steps=[('vectorizer',
                 TfidfVectorizer(analyzer='word', binary=False,
                                 decode_error='strict',
                                 dtype=<class 'numpy.float64'>,
                                 encoding='utf-8', input='content',
                                 lowercase=True, max_df=1.0, max_features=None,
                                 min_df=10, ngram_range=(1, 2), norm='l2',
                                 preprocessor=None, smooth_idf=True,
                                 stop_words=None, strip_accents=None,
                                 sublinear_tf=False,
                                 token_pattern='(?u)\\b\\w\\w+\\b',
                                 tokenizer=None, use_idf=True,
                                 vocabulary=None)),
                ('model',
                 LogisticRegression(C=10, class_weight=None, dual=False,
                                    fit_intercept=True, intercept_scaling=1,
                                    l1_ratio=None, max_iter=200,
                                    multi_class='auto', n_jobs=None,
                                    penalty='l2', random_state=None,
                                    solver='lbfgs', tol=0.0001, verbose=0,
                                    warm_start=False))],
         verbose=False)
```

In [28]:
```
grid_search.best_params_
```

Out[28]:
```
{'model__C': 10, 'vectorizer__ngram_range': (1, 2)}
```

In [ ]:
```
plot_roc_curve(grid_search.best_estimator_, X_test, y_test)
```