Cameron Lee
ML Milestone 3

Dataset: Drug Review

Problem Statement
The problem statement that I am proposing is classifying the effectiveness of a drug. Accurately classifying the effectiveness of drugs is crucial for medical professionals, researchers, and patients to make informed decisions regarding treatment options. By classifying the effectiveness of these drugs, we can provide valuable insights into their efficacy, enabling healthcare providers to prescribe the most suitable medications, researchers to identify trends in drug performance, and patients to make well-informed decisions about their healthcare.

Solution
The problem statement that I am proposing is classifying the effectiveness of a drug. Therefore, effectiveness will be used as the target. In particular, I created a binary classification with the target focused on whether the drug is considered "effective" or "ineffective". These are the given features I will be focusing on to drive the study

urlDrugName	effectiveness	sideEffects	condition	rating	benefits Review	sideEffects Review	comments Review

In addition to those features, I implemented text preprocessing using the NLTK library for the reviews and performed sentiment analysis using the TextBlob library. These sentiment analysis fields show negative values for negative sentiment analysis and positive values for positive sentiment analysis. These columns were added to the test data with the following names:

sideEffectsSentiment	commentsSentiment	benefitsSentiment	avgSentiment

I took the average score of sideEffectsSentiment, commentsSentiment, and benefitsSentiment and rounded the values to create a column called "avgSentiment" which corresponds to the average sentiment from all three types of reviews present in each datapoint.

I made the design decision for an "effective" classification, the data would have to meet the target criteria of the following:

```
targetTrain = (((train_imp["effectiveness"] == "Highly Effective") |
                (train_imp["effectiveness"] == "Considerably Effective")) &
                (train_imp["avgSentiment"] >= 0))
```

If this criteria is not met, the classification of that datapoint would be considered "ineffective"
After a long process of data preprocessing and feature engineering that included text preprocessing, sentiment analysis implementation, handling missing values, encoding categorical columns, dropping unneeded columns, and splitting the datasets into their respective sets (assigning target values, test and train datasets were given), I was ready to begin classifiers to help me solve the problem.

In order to address the classification problem I used the three following classifiers:

Random Forest
I started by developing a Random Forest model as I have found that it works well by combining the power of multiple decision trees, leading to robust and accurate decisions. I began by using Randomized Search Cross validation with a broad set of hyperparameters in order to narrow down the range of which hyperparameters were the most relevant. This yielded the following:

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
{'n_estimators': 400, 'min_samples_split': 10, 'min_samples_leaf': 1, 'max_features': 'sqrt', 'max_depth': 20, 'bootstrap': True}
Random Forest Classifier Metrics:
Testing Accuracy:  0.6003861003861004
```

From there, I narrowed down the values and switched to GridSearch cross validation.

```
param_grid = {
    'bootstrap': [True],
    'max_depth': [30, 40, 50, 100],
    'max_features': ['sqrt'],
    'min_samples_leaf': [1, 2, 3],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [350, 375, 400, 425, 450]
}
```

Then I ran the entire algorithm on the updated y_pred and tested it against the Y_test (target value I defined). This yielded the following:

```
Random Forest Classifier Metrics:
Accuracy: 0.8040540540540541
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.89      0.85       625
           1       0.80      0.67      0.73       411

    accuracy                           0.80      1036
   macro avg       0.80      0.78      0.79      1036
weighted avg       0.80      0.80      0.80      1036
```

At this point in time, I was testing the effectiveness of sentiment analysis. By including the average sentiment analysis on top of the values for each type of review already present, I saw a 5 percent increase in accuracy. This was the final report for Random Forest.

```
Random Forest Classifier Metrics:
Accuracy: 0.8455598455598455
Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.80      0.76       318
           1       0.91      0.87      0.89       718

    accuracy                           0.85      1036
   macro avg       0.82      0.83      0.82      1036
weighted avg       0.85      0.85      0.85      1036
```

<u>Gaussian Naive Bayes</u>

I chose to use Naive Bayes due to the ability to create a strong distinction between classifications, specifically an “effective” review and an “ineffective”. As the dataset has a lot of text data, it made sense for me to include those metrics such as sentiment analysis and look for positive and negative results.

With this model, I had prepared the train and test datasets with the set targets. I then fit the data, predicted the values and this gave me the following:

```
GNB Classifier Metrics:
Accuracy: 0.7268339768339769
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.66      0.74       625
           1       0.61      0.83      0.71       411

    accuracy                           0.73      1036
   macro avg       0.74      0.75      0.73      1036
weighted avg       0.76      0.73      0.73      1036
```

Through my research, I found there was not much tuning I could do with this model due to its simplistic nature. So I applied ‘smoothing’ to account for any zero probabilities and also included the average sentiment since I found success with it in Random Forest and saw about a 10 percent increase in accuracy. Here is the final report for Guassian Naive Bayes:

```
GNB Classifier Metrics:
Accuracy: 0.8185328185328186
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.61      0.67       318
           1       0.84      0.91      0.87       718

    accuracy                           0.82      1036
   macro avg       0.80      0.76      0.77      1036
weighted avg       0.81      0.82      0.81      1036
```

Support Vector Machine

Finally for this model, I had prepared the train and test datasets with the set targets. I fit the data, predicted the values and this gave me the following:

```
SVM Classifier Metrics:
Accuracy: 0.6032818532818532
Classification Report:
              precision    recall  f1-score   support

           0       0.60      1.00      0.75       625
           1       0.00      0.00      0.00       411

    accuracy                           0.60      1036
   macro avg       0.30      0.50      0.38      1036
weighted avg       0.36      0.60      0.45      1036
```

This was the lowest accuracy I had produced and since this was the last model I created, I felt that I had exhausted the possibility to improve the dataset in the preprocessing stage. I tuned the C and Gamma values of the SVC model, selected the best parameters and narrowed down the hyperparameters. This produced the final report for SVM with an increase of 9 percent accuracy.

```
SVM Classifier Metrics:
Accuracy: 0.693050193050193
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       318
           1       0.69      1.00      0.82       718

    accuracy                           0.69      1036
   macro avg       0.35      0.50      0.41      1036
weighted avg       0.48      0.69      0.57      1036
```

Training and Testing Dataset

The training and testing datasets were provided in the database. The following gives insight into the size of each respective dataset. After further analysis, this is about a 1:3 ratio of test data to train data. As mentioned above, the targets were set by a predefined “effectiveness” classification.

	X_train	Y_train	X_test	Y_test
# Entries	3107	3107	1036	1036

Results

By using accuracy as the main metric for the accuracy of classification of each of the models, here is the final result.

	RF	GNB	SVM
Accuracy	0.85	0.82	0.69

Random Forest was able to produce the best accuracy rate in classifying the effectiveness of the drugs. GNB was also effective in doing so with the implementation of smoothing. I believe that GNB gave great insight into the classification of this data and perhaps with additional feature engineering could prove to be very effective due to the simplicity and computation speed. Random Forest required extensive tuning compared to the rest of the models and computation time was the longest. SVM produced the worst accuracy. However with additional feature engineering along with tuning the weights of that additional data, could yield a greater accuracy.

If I were to take this investigation further, I would look to create additional sources of analysis to find strong correlations between certain features and perform additional feature engineering. I believe the models can be improved with additional data with careful attention to avoid overfitting or underfitting.

In conclusion, I was able to classify the effectiveness of drugs in this dataset by defining a target that the models would be looking to meet, implementing Random Forest, Gaussian Naive Bayes and Support Vector Machine, and analyzing the classification report.

Code: (Not intended to be part of the report with a 3 page limit, just wanted to include at the end for additional insight)

```python
import pandas as pd
import numpy as np

### Data Preprocessing ###
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
#sentiment analysis
from textblob import TextBlob
#text processing
import re
import nltk

### Hyperparameter tuning ###
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

### models ###
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB

### Evaluation ###
from sklearn.metrics import (
    accuracy_score,
    classification_report
)

####### READING IN DATA & PREPROCESSING #######
dtypes = {
    "Unnamed: 0": "numerical",
    "urlDrugName": "categorical",
    "rating": "numerical",
    "effectiveness": "categorical",
    "sideEffects": "categorical",
    "condition": "categorical",
    "benefitsReview": "text",
    "sideEffectsReview": "text",
    "commentsReview": "text",
}

#Load training data
train_data = pd.read_csv('drugLib_raw/drugLibTrain_raw.tsv', sep='\t')
#Load test data
test_data = pd.read_csv('drugLib_raw/drugLibTest_raw.tsv', sep='\t')


### Review Preprocessing (for text values) ###
def utils_preprocess_text(text, flg_stemm=False, flg_lemm=True, lst_stopwords=None):
    ## clean (convert to lowercase and remove punctuations and characters and then strip)
    text = re.sub(r'[^\w\s]', '', str(text).lower().strip())

    ## Tokenize (convert from string to list)
    lst_text = text.split()
    ## remove Stopwords
    if lst_stopwords is not None:
        lst_text = [word for word in lst_text if word not in
                    lst_stopwords]

    ## Stemming (remove -ing, -ly, ...)
    if flg_stemm == True:
        ps = nltk.stem.porter.PorterStemmer()
        lst_text = [ps.stem(word) for word in lst_text]

    ## Lemmatisation (convert the word into root word)
    if flg_lemm == True:
        lem = nltk.stem.wordnet.WordNetLemmatizer()
        lst_text = [lem.lemmatize(word) for word in lst_text]

    ## back to string from list
    text = " ".join(lst_text)
    return text

### Clean for sentiment analysis ###
train_data["sideEffectsReview"] = train_data["sideEffectsReview"].apply(lambda x: utils_preprocess_text(x, flg_stemm=False, flg_lemm=True))
train_data["commentsReview"] = train_data["commentsReview"].apply(lambda x: utils_preprocess_text(x, flg_stemm=False, flg_lemm=True))
train_data["benefitsReview"] = train_data["benefitsReview"].apply(lambda x: utils_preprocess_text(x, flg_stemm=False, flg_lemm=True))


train_data['sideEffectsReview'] = train_data['sideEffectsReview'].astype(str)
train_data["sideEffectsSentiment"] = train_data["sideEffectsReview"].apply(lambda x: TextBlob(x).sentiment.polarity)

train_data['commentsReview'] = train_data['commentsReview'].astype(str)
train_data["commentsSentiment"] = train_data["commentsReview"].apply(lambda x: TextBlob(x).sentiment.polarity)


train_data['benefitsReview'] = train_data['benefitsReview'].astype(str)
train_data["benefitsSentiment"] = train_data["benefitsReview"].apply(lambda x: TextBlob(x).sentiment.polarity)

##################

test_data["sideEffectsReview"] = test_data["sideEffectsReview"].apply(lambda x: utils_preprocess_text(x, flg_stemm=False, flg_lemm=True))
test_data["commentsReview"] = test_data["commentsReview"].apply(lambda x: utils_preprocess_text(x, flg_stemm=False, flg_lemm=True))
test_data["benefitsReview"] = test_data["benefitsReview"].apply(lambda x: utils_preprocess_text(x, flg_stemm=False, flg_lemm=True))


test_data['sideEffectsReview'] = test_data['sideEffectsReview'].astype(str)
test_data["sideEffectsSentiment"] = test_data["sideEffectsReview"].apply(lambda x: TextBlob(x).sentiment.polarity)

test_data['commentsReview'] = test_data['commentsReview'].astype(str)
test_data["commentsSentiment"] = test_data["commentsReview"].apply(lambda x: TextBlob(x).sentiment.polarity)


test_data['benefitsReview'] = test_data['benefitsReview'].astype(str)
test_data["benefitsSentiment"] = test_data["benefitsReview"].apply(lambda x: TextBlob(x).sentiment.polarity)
```

```
##################################################

#overall sentiment classificaiton
train_data["avgSentiment"] = (train_data["sideEffectsSentiment"]+train_data["sideEffectsSentiment"]+train_data["sideEffectsSentiment"]) / 3
train_data["avgSentiment"] = round(train_data["avgSentiment"])

test_data["avgSentiment"] = (test_data["sideEffectsSentiment"]+test_data["sideEffectsSentiment"]+test_data["sideEffectsSentiment"]) / 3
test_data["avgSentiment"] = round(test_data["avgSentiment"])

## Handling the missing values and assigning old column names
train_imp, test_imp = [
    pd.DataFrame(
        SimpleImputer(strategy="most_frequent").fit_transform(df), columns=df.columns
    )
    for df in (train_data, test_data)
]

## Encoding the categorical columns
for i in ["urlDrugName", "condition", "sideEffects"]:
    train_imp[i] = LabelEncoder().fit_transform(train_imp[i])
    test_imp[i] = LabelEncoder().fit_transform(test_imp[i])

### drop text columns
train_imp.drop("benefitsReview", axis=1, inplace=True)
train_imp.drop("sideEffectsReview", axis=1, inplace=True)
train_imp.drop("commentsReview", axis=1, inplace=True)

test_imp.drop("benefitsReview", axis=1, inplace=True)
test_imp.drop("sideEffectsReview", axis=1, inplace=True)
test_imp.drop("commentsReview", axis=1, inplace=True)

train_imp.columns = train_imp.columns.astype(str)
test_imp.columns = test_imp.columns.astype(str)

#determine target based off of effectiveness and sentiment
#Design decision: if highly effective or considerably effective and has
#positive sentiment or neutral sentiment present -> effective classification
#else -> ineffective classification

targetTrain = (((train_imp["effectiveness"] == "Highly Effective") |
                      (train_imp["effectiveness"] == "Considerably Effective")) &
                      (train_imp["avgSentiment"] >= 0))

targetTest = (((test_imp["effectiveness"] == "Highly Effective") |
                      (test_imp["effectiveness"] == "Considerably Effective")) &
                      (test_imp["avgSentiment"] >= 0))

train_imp["target"] = (targetTrain).astype(int)
test_imp["target"] = (targetTest).astype(int)


## Splitting the train and test datasets into feature variables
X_train, Y_train = train_imp.drop(["effectiveness", "target"], axis=1), train_imp["target"]
X_test, Y_test = test_imp.drop(["effectiveness", "target"], axis=1), test_imp["target"]
##########################################################################
```

```
159
160 ####### Models (Fitting, Tuning, Evaluation) #######
161
162 def print_metrics(y_true, y_pred):
163     # Accuracy
164     accuracy = accuracy_score(y_true, y_pred)
165     print("Accuracy:", accuracy)
166
167     # Classification Report
168     class_report = classification_report(y_true, y_pred)
169     print("Classification Report:")
170     print(class_report)
171
172
173 ####### SVM #######
174 svm_clf = svm.SVC()
175
176 param_grid = {'C': [0.1, 1, 2],
177               'gamma': [1, 0.1, 0.01],
178               'kernel': ['rbf']}
179
180
181 grid = GridSearchCV(svm_clf, param_grid, refit = True, verbose = 3, n_jobs=-1)
182
183 grid.fit(X_train,Y_train)
184
185 best_model = grid.best_estimator_
186
187 svm_y_pred = best_model.predict(X_test)
188
189 print("SVM Classifier Metrics:")
190 print_metrics(Y_test, svm_y_pred)
191
192 ####### GNB #######
193 gnb = GaussianNB()
194
195 #Apply smoothing param
196 params_NB = {'var_smoothing': np.logspace(0,-9, num=100)}
197 gs_NB = GridSearchCV(estimator=gnb,
198                  param_grid=params_NB,
199                  cv=3,
200                  verbose=1,
201                  scoring='accuracy')
202
203 gs_NB.fit(X_train,Y_train)
204
205 best_params = gs_NB.best_params_
206
207 best_model = gs_NB.best_estimator_
208
209 gnb_y_pred = best_model.predict(X_test)
210 print("GNB Classifier Metrics:")
211 print_metrics(Y_test, gnb_y_pred)
212
213 ####### Random Forest #######
214 param_grid = {
215     'bootstrap': [True],
216     'max_depth': [30, 40, 50, 100],
217     'max_features': ['sqrt'],
218     'min_samples_leaf': [1, 2, 3],
219     'min_samples_split': [8, 10, 12],
220     'n_estimators': [350, 375, 400, 425, 450]
221 }
222
223 rf = RandomForestClassifier()
224
225 #Tune Hyperparameters
226 rand_rf = GridSearchCV(estimator = rf, param_grid = param_grid, cv = 3, verbose=2, n_jobs = -1)
227
228 rand_rf.fit(X_train, Y_train)
229 best_params = rand_rf.best_params_
230 best_model = rand_rf.best_estimator_
231
232 rf_y_pred = best_model.predict(X_test)
233
234 print("Random Forest Classifier Metrics:")
235 print_metrics(Y_test, rf_y_pred)
236 ###############################################################################
```