Text Classification Assessment

This assessment is very much like the Text Classification Project we just completed, and the dataset is very similar.

The **moviereviews2.tsv** dataset contains the text of 6000 movie reviews. 3000 are positive, 3000 are negative, and the text has been preprocessed as a tab-delimited file. As before, labels are given as `pos` and `neg`.

We've included 20 reviews that contain either `NaN` data, or have strings made up of whitespace.

For more information on this dataset visit http://ai.stanford.edu/~amaas/data/sentiment/

Task #1: Perform imports and load the dataset into a pandas DataFrame

For this exercise you can load the dataset from `'../TextFiles/moviereviews2.tsv'`.

In [1]:
```
import pandas as pd

df = pd.read_csv("moviereviews2.tsv", sep="\t")
df.head()
```

Out[1]:

	label	review
0	pos	I loved this movie and will watch it again. Or...
1	pos	A warm, touching movie that has a fantasy-like...
2	pos	I was not expecting the powerful filmmaking ex...
3	neg	This so-called "documentary" tries to tell tha...
4	pos	This show has been my escape from reality for ...

Task #2: Check for missing values:

In [2]:
```
# Check for NaN values:
df.isna().sum()
```

Out[2]:
```
label      0
review    20
dtype: int64
```

In [3]:
```
# Check for whitespace strings (it's OK if there aren't any!):
blank_index = []

for index, label, review in df.itertuples():
    if type(review) == "str":
            if review.isspace():
                blank_index.append(index)
blank_index
```

Out[3]: []

Task #3: Remove NaN values:

In [4]:
```
df = df.dropna(axis=0, how="any", subset=["review"])
df.isna().sum()
```

Out[4]:
```
label     0
review    0
dtype: int64
```

Task #4: Take a quick look at the `label` column:

In [5]:
```
df["label"].value_counts()
```

Out[5]:
```
neg    2990
pos    2990
Name: label, dtype: int64
```

Task #5: Split the data into train & test sets:

You may use whatever settings you like. To compare your results to the solution notebook, use `test_size=0.33, random_state=42`

In [6]:
```
X = df["review"]
y = df["label"]
```

In [7]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

Task #6: Build a pipeline to vectorize the date, then train and fit a model

You may use whatever model you like. To compare your results to the solution notebook, use `LinearSVC`.

In [8]:
```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

# Linear SVC:
text_clf_lsvc = Pipeline([("tfidf", TfidfVectorizer()),
                          ("clf_lsvc", LinearSVC())])

# Naïve Bayes:
text_clf_nb = Pipeline([("tfidf", TfidfVectorizer()),
                        ("clf_nb", MultinomialNB())])

# Logistic Regression
text_clf_lr = Pipeline([("tfidf", TfidfVectorizer()),
                        ("clf_lr", LogisticRegression())])

# KNN
text_clf_knn = Pipeline([("tfidf", TfidfVectorizer()),
                         ("clf_knn", KNeighborsClassifier(n_neighbors=2))])

# Decession Tree
text_clf_dtree = Pipeline([("tfidf", TfidfVectorizer()),
                        ("clf_lr", DecisionTreeClassifier())])

# Random Forest
text_clf_rfc = Pipeline([("tfidf", TfidfVectorizer()),
                        ("clf_lr", RandomForestClassifier())])
```

Task #7: Run predictions and analyze the results

In [9]:
```
# Form a prediction set

# Linear SVC:
text_clf_lsvc.fit(X_train, y_train)
predictions_lsvc = text_clf_lsvc.predict(X_test)

# Naïve Bayes:
text_clf_nb.fit(X_train, y_train)
predictions_nb = text_clf_nb.predict(X_test)

# Logistic Regression
text_clf_lr.fit(X_train, y_train)
predictions_lr = text_clf_lr.predict(X_test)

# KNN
text_clf_knn.fit(X_train, y_train)
predictions_knn = text_clf_knn.predict(X_test)

# Decession Tree
text_clf_dtree.fit(X_train, y_train)
predictions_dtree = text_clf_dtree.predict(X_test)

# Random Forest
text_clf_rfc.fit(X_train, y_train)
predictions_rfc = text_clf_rfc.predict(X_test)
```

In [10]:
```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
# Report the confusion matrix


# Linear SVC:
print(f"# Linear SVC #\n{confusion_matrix(y_test, predictions_lsvc)}")

# Naïve Bayes:
print(f"# Naïve Bayes #\n{confusion_matrix(y_test, predictions_nb)}")

# Logistic Regression
print(f"# Logistic Regression #\n{confusion_matrix(y_test, predictions_lr)}")

# KNN
print(f"# KNN #\n{confusion_matrix(y_test, predictions_knn)}")

# Decession Tree
print(f"# Decession Tree #\n{confusion_matrix(y_test, predictions_dtree)}")

# Random Forest
print(f"# Random Forest #\n{confusion_matrix(y_test, predictions_rfc)}")
```

```
# Linear SVC #
[[900  91]
 [ 63 920]]
# Naïve Bayes #
[[940  51]
 [136 847]]
# Logistic Regression #
```

```
[[887 104]
 [ 76 907]]
# KNN #
[[893  98]
 [371 612]]
# Decession Tree #
[[745 246]
 [251 732]]
# Random Forest #
[[877 114]
 [117 866]]
```

In [11]:

```
# Print a classification report

# Linear SVC:
print(f"# Linear SVC #\n{classification_report(y_test, predictions_lsvc)}")

# Naïve Bayes:
print(f"# Naïve Bayes #\n{classification_report(y_test, predictions_nb)}")

# Logistic Regression
print(f"# Logistic Regression #\n{classification_report(y_test, predictions_lr)}")

# KNN
print(f"# KNN #\n{classification_report(y_test, predictions_knn)}")

# Decession Tree
print(f"# Decession Tree #\n{classification_report(y_test, predictions_dtree)}")

# Random Forest
print(f"# Random Forest #\n{classification_report(y_test, predictions_rfc)}")
```

```
# Linear SVC #
              precision    recall  f1-score   support

         neg       0.93      0.91      0.92       991
         pos       0.91      0.94      0.92       983

    accuracy                           0.92      1974
   macro avg       0.92      0.92      0.92      1974
weighted avg       0.92      0.92      0.92      1974

# Naïve Bayes #
              precision    recall  f1-score   support

         neg       0.87      0.95      0.91       991
         pos       0.94      0.86      0.90       983

    accuracy                           0.91      1974
   macro avg       0.91      0.91      0.91      1974
weighted avg       0.91      0.91      0.91      1974

# Logistic Regression #
              precision    recall  f1-score   support

         neg       0.92      0.90      0.91       991
         pos       0.90      0.92      0.91       983

    accuracy                           0.91      1974
   macro avg       0.91      0.91      0.91      1974
weighted avg       0.91      0.91      0.91      1974

# KNN #
              precision    recall  f1-score   support

         neg       0.71      0.90      0.79       991
         pos       0.86      0.62      0.72       983

    accuracy                           0.76      1974
   macro avg       0.78      0.76      0.76      1974
weighted avg       0.78      0.76      0.76      1974

# Decession Tree #
              precision    recall  f1-score   support

         neg       0.75      0.75      0.75       991
         pos       0.75      0.74      0.75       983

    accuracy                           0.75      1974
   macro avg       0.75      0.75      0.75      1974
weighted avg       0.75      0.75      0.75      1974

# Random Forest #
              precision    recall  f1-score   support

         neg       0.88      0.88      0.88       991
         pos       0.88      0.88      0.88       983

    accuracy                           0.88      1974
   macro avg       0.88      0.88      0.88      1974
weighted avg       0.88      0.88      0.88      1974
```

In [12]:

```
# Print the overall accuracy

# Linear SVC:
print(f"# Linear SVC #\n{accuracy_score(y_test, predictions_lsvc)}")

# Naïve Bayes:
print(f"# Naïve Bayes #\n{accuracy_score(y_test, predictions_nb)}")
```

```
# Logistic Regression
print(f"# Logistic Regression #\n{accuracy_score(y_test, predictions_lr)}")

# KNN
print(f"# KNN #\n{accuracy_score(y_test, predictions_knn)}")

# Decession Tree
print(f"# Decession Tree #\n{accuracy_score(y_test, predictions_dtree)}")

# Random Forest
print(f"# Random Forest #\n{accuracy_score(y_test, predictions_rfc)}")
```

```
# Linear SVC #
0.9219858156028369
# Naïve Bayes #
0.9052684903748733
# Logistic Regression #
0.9088145896656535
# KNN #
0.7624113475177305
# Decession Tree #
0.74822695035461
# Random Forest #
0.8829787234042553
```

Great job!

In [13]:
```
myreview = "A movie I really wanted to love was terrible. \
I'm sure the producers had the best intentions, but the execution was lacking."
```

In [14]:
```
text_clf_lsvc.predict(["This is my test!"])
```

Out[14]:
```
array(['pos'], dtype=object)
```

In [15]:
```
text_clf_lsvc.predict([myreview])
```

Out[15]:
```
array(['neg'], dtype=object)
```

In [16]:
```
text_clf_nb.predict([myreview])
```

Out[16]:
```
array(['neg'], dtype='<U3')
```

In [17]:
```
text_clf_lr.predict([myreview])
```

Out[17]:
```
array(['neg'], dtype=object)
```

In [18]:
```
text_clf_knn.predict([myreview])
```

Out[18]:
```
array(['neg'], dtype=object)
```

In [19]:
```
text_clf_dtree.predict([myreview])
```

Out[19]:
```
array(['neg'], dtype=object)
```

In [20]:
```
text_clf_rfc.predict([myreview])
```

Out[20]:
```
array(['neg'], dtype=object)
```