Import necessary depencencies

In [1]:

```
import pandas as pd
import numpy as np
import text_normalizer as tn
import model_evaluation_utils as meu
import spacy

np.set_printoptions(precision=2, linewidth=80)
```

Load and normalize data

In [3]:

```
dataset = pd.read_csv(r'movie_reviews_cleaned.csv')
# take a peek at the data
print(dataset.head())
reviews = np.array(dataset['review'])
sentiments = np.array(dataset['sentiment'])

# build train and test datasets
norm_train_reviews = reviews[:35000]
norm_train_sentiments = sentiments[:35000]
norm_test_reviews = reviews[35000:]
norm_test_sentiments = sentiments[35000:]
```

```
                                              review sentiment
0  not bother think would see movie great supspen...  negative
1  careful one get mitt change way look kung fu f...  positive
2  chili palmer tired movie know want success mus...  negative
3  follow little know 1998 british film make budg...  positive
4  dark angel cross huxley brave new world percys...  positive
```

Traditional Supervised Machine Learning Models

Feature Engineering

In [4]:

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

# build BOW features on train reviews
cv = CountVectorizer(binary=False, min_df=0.0, max_df=1.0, ngram_range=(1,2))
cv_train_features = cv.fit_transform(norm_train_reviews)
# build TFIDF features on train reviews
tv = TfidfVectorizer(use_idf=True, min_df=0.0, max_df=1.0, ngram_range=(1,2),
                     sublinear_tf=True)
tv_train_features = tv.fit_transform(norm_train_reviews)
```

In [5]:

```
# transform test reviews into features
cv_test_features = cv.transform(norm_test_reviews)
tv_test_features = tv.transform(norm_test_reviews)
```

In [6]:

```
print('BOW model:> Train features shape:', cv_train_features.shape, ' Test features
print('TFIDF model:> Train features shape:', tv_train_features.shape, ' Test feature
```

```
BOW model:> Train features shape: (35000, 2099202)  Test features shap
e: (15000, 2099202)
TFIDF model:> Train features shape: (35000, 2099202)  Test features sh
ape: (15000, 2099202)
```

Model Training, Prediction and Performance Evaluation

In [7]:

```
from sklearn.linear_model import SGDClassifier, LogisticRegression

lr = LogisticRegression(penalty='l2', max_iter=100, C=1)
svm = SGDClassifier(loss='hinge', n_iter=100)
```

In [10]:

```
# Logistic Regression model on BOW features
lr_bow_predictions = meu.train_predict_model(classifier=lr,
                                             train_features=cv_train_features, train
                                             test_features=cv_test_features, test_la
meu.display_model_performance_metrics(true_labels=norm_test_sentiments, predicted_la
                                      classes=['positive', 'negative'])
```

```
Model Performance metrics:
------------------------------
Accuracy: 0.8985
Precision: 0.8985
Recall: 0.8985
F1 Score: 0.8985

Model Classification report:
------------------------------
             precision    recall  f1-score   support

   positive       0.89      0.91      0.90      7587
   negative       0.90      0.89      0.90      7413

avg / total       0.90      0.90      0.90     15000


Prediction Confusion Matrix:
------------------------------
                 Predicted:
                   positive negative
Actual: positive       6873      714
        negative        809     6604
```

In [12]:

```
# Logistic Regression model on TF-IDF features
lr_tfidf_predictions = meu.train_predict_model(classifier=lr,
                                               train_features=tv_train_features, tra
                                               test_features=tv_test_features, test_
meu.display_model_performance_metrics(true_labels=norm_test_sentiments, predicted_la
                                      classes=['positive', 'negative'])
```

```
Model Performance metrics:
------------------------------
Accuracy: 0.8919
Precision: 0.8921
Recall: 0.8919
F1 Score: 0.8919

Model Classification report:
------------------------------
             precision    recall  f1-score   support

   positive       0.89      0.90      0.89      7587
   negative       0.90      0.88      0.89      7413

avg / total       0.89      0.89      0.89     15000


Prediction Confusion Matrix:
------------------------------
```

In [13]:

```
svm_bow_predictions = meu.train_predict_model(classifier=svm,
                                              train_features=cv_train_features, train
                                              test_features=cv_test_features, test_la
meu.display_model_performance_metrics(true_labels=norm_test_sentiments, predicted_la
                                      classes=['positive', 'negative'])
```

```
/Users/james/anaconda3/lib/python3.6/site-packages/sklearn/linear_mode
l/stochastic_gradient.py:117: DeprecationWarning: n_iter parameter is
deprecated in 0.19 and will be removed in 0.21. Use max_iter and tol i
nstead.
  DeprecationWarning)

Model Performance metrics:
------------------------------
Accuracy: 0.8985
Precision: 0.8988
Recall: 0.8985
F1 Score: 0.8985

Model Classification report:
------------------------------
             precision    recall  f1-score   support

   positive       0.89      0.91      0.90      7587
   negative       0.91      0.88      0.90      7413

avg / total       0.90      0.90      0.90     15000


Prediction Confusion Matrix:
------------------------------
                  Predicted:
                    positive negative
Actual: positive        6921      666
        negative         856     6557
```

In [15]:

```
svm_tfidf_predictions = meu.train_predict_model(classifier=svm,
                                                train_features=tv_train_features, tr
                                                test_features=tv_test_features, test
meu.display_model_performance_metrics(true_labels=norm_test_sentiments, predicted_la
                                      classes=['positive', 'negative'])
```

```
/Users/james/anaconda3/lib/python3.6/site-packages/sklearn/linear_mode
l/stochastic_gradient.py:117: DeprecationWarning: n_iter parameter is
deprecated in 0.19 and will be removed in 0.21. Use max_iter and tol i
nstead.
  DeprecationWarning)

Model Performance metrics:
------------------------------
Accuracy: 0.8953
Precision: 0.8957
Recall: 0.8953
F1 Score: 0.8953

Model Classification report:
------------------------------
             precision    recall  f1-score   support

   positive       0.88      0.91      0.90      7587
   negative       0.91      0.88      0.89      7413

avg / total       0.90      0.90      0.90     15000


Prediction Confusion Matrix:
------------------------------
                  Predicted:
                    positive negative
Actual: positive        6916      671
        negative         899     6514
```

Newer Supervised Deep Learning Models

In [ ]:

```
import gensim
import keras
from keras.models import Sequential
from keras.layers import Dropout, Activation, Dense
from sklearn.preprocessing import LabelEncoder
```

Prediction class label encoding

In [ ]:

```
le = LabelEncoder()
num_classes=2
# tokenize train reviews & encode train labels
tokenized_train = [tn.tokenizer.tokenize(text)
                   for text in norm_train_reviews]
y_tr = le.fit_transform(norm_train_sentiments)
y_train = keras.utils.to_categorical(y_tr, num_classes)
# tokenize test reviews & encode test labels
tokenized_test = [tn.tokenizer.tokenize(text)
                   for text in norm_test_reviews]
y_ts = le.fit_transform(norm_train_sentiments)
y_test = keras.utils.to_categorical(y_ts, num_classes)
```

In [ ]:

```
# print class label encoding map and encoded labels
print('Sentiment class label map:', dict(zip(le.classes_, le.transform(le.classes_))
print('Sample test label transformation:\n'+'-'*35,
      '\nActual Labels:', norm_test_sentiments[:3], '\nEncoded Labels:', y_ts[:3],
      '\nOne hot encoded Labels:\n', y_test[:3])
```

Feature Engineering with word embeddings

In [ ]:

```
# build word2vec model
w2v_num_features = 500
w2v_model = gensim.models.Word2Vec(tokenized_train, size=w2v_num_features, window=15
                                   min_count=10, sample=1e-3)
```

In [ ]:

```
def averaged_word2vec_vectorizer(corpus, model, num_features):
    vocabulary = set(model.wv.index2word)

    def average_word_vectors(words, model, vocabulary, num_features):
        feature_vector = np.zeros((num_features,), dtype="float64")
        nwords = 0.

        for word in words:
            if word in vocabulary:
                nwords = nwords + 1.
                feature_vector = np.add(feature_vector, model[word])
        if nwords:
            feature_vector = np.divide(feature_vector, nwords)

        return feature_vector

    features = [average_word_vectors(tokenized_sentence, model, vocabulary, num_feat
                    for tokenized_sentence in corpus]
    return np.array(features)
```

In [ ]:

```
# generate averaged word vector features from word2vec model
avg_wv_train_features = averaged_word2vec_vectorizer(corpus=tokenized_train, model=w
                                                     num_features=500)
avg_wv_test_features = averaged_word2vec_vectorizer(corpus=tokenized_test, model=w2v
                                                    num_features=500)
```

In [ ]:

```
# feature engineering with GloVe model
train_nlp = [tn.nlp(item) for item in norm_train_reviews]
train_glove_features = np.array([item.vector for item in train_nlp])

test_nlp = [tn.nlp(item) for item in norm_test_reviews]
test_glove_features = np.array([item.vector for item in test_nlp])
```

In [ ]:

```
print('Word2Vec model:> Train features shape:', avg_wv_train_features.shape, ' Test
print('GloVe model:> Train features shape:', train_glove_features.shape, ' Test feat
```

Modeling with deep neural networks

Building Deep neural network architecture

In [ ]:

```
def construct_deepnn_architecture(num_input_features):
    dnn_model = Sequential()
    dnn_model.add(Dense(512, activation='relu', input_shape=(num_input_features,)))
    dnn_model.add(Dropout(0.2))
    dnn_model.add(Dense(512, activation='relu'))
    dnn_model.add(Dropout(0.2))
    dnn_model.add(Dense(512, activation='relu'))
    dnn_model.add(Dropout(0.2))
    dnn_model.add(Dense(2))
    dnn_model.add(Activation('softmax'))

    dnn_model.compile(loss='categorical_crossentropy', optimizer='adam',
                      metrics=['accuracy'])
    return dnn_model
```

In [ ]:

```
w2v_dnn = construct_deepnn_architecture(num_input_features=500)
```

Visualize sample deep architecture

In [ ]:

```
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot

SVG(model_to_dot(w2v_dnn, show_shapes=True, show_layer_names=False,
                 rankdir='TB').create(prog='dot', format='svg'))
```

Model Training, Prediction and Performance Evaluation

In [ ]:

```
batch_size = 100
w2v_dnn.fit(avg_wv_train_features, y_train, epochs=5, batch_size=batch_size,
            shuffle=True, validation_split=0.1, verbose=1)
```

In [ ]:

```
y_pred = w2v_dnn.predict_classes(avg_wv_test_features)
predictions = le.inverse_transform(y_pred)
```

In [ ]:

```
meu.display_model_performance_metrics(true_labels=test_sentiments, predicted_labels=
                                      classes=['positive', 'negative'])
```

In [ ]:

```
glove_dnn = construct_deepnn_architecture(num_input_features=300)
```

In [ ]:

```
batch_size = 100
glove_dnn.fit(train_glove_features, y_train, epochs=5, batch_size=batch_size,
              shuffle=True, validation_split=0.1, verbose=1)
```

In [ ]:

```
y_pred = glove_dnn.predict_classes(test_glove_features)
predictions = le.inverse_transform(y_pred)
```

In [ ]:

```
meu.display_model_performance_metrics(true_labels=test_sentiments, predicted_labels=
                                      classes=['positive', 'negative'])
```