Import necessary depencencies

In [1]:
```
ENABLE_COLAB=False
```

In [2]:
```
if ENABLE_COLAB:
  !pip install pycaret -q
  #!pip install https://github.com/pandas-profiling/pandas-profiling/archive/
  #!pip install matplotlib -q
  #!pip install pandasql -q
else:
  display('Google Colab not enabled')
```

```
'Google Colab not enabled'
```

In [3]:
```
if ENABLE_COLAB:
  from pycaret.utils import enable_colab
  enable_colab()

  from google.colab import drive
  drive.mount('/content/gdrive', force_remount=True)
else:
  display('Google Colab not enabled')
```

```
'Google Colab not enabled'
```

In [5]:
```
import os
import sys
print("Current working directory: {0}".format(os.getcwd()))
os.chdir('/home/magni/ML_Root/project_root/utility_files')
print("Current working directory: {0}".format(os.getcwd()))
sys.path.append('.')
```

```
Current working directory: /home/magni/ML_Root/project_root/ML1010_Weekly
Current working directory: /home/magni/ML_Root/project_root/utility_files
```

In [6]:
```
import model_evaluation_utils as meu
```

In [7]:
```
import pandas as pd
import numpy as np

#import text_normalizer as tn


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

Load and normalize data

In [9]:

```
dataset = pd.read_csv('/home/magni/ML_Root/project_root/data/ML1010_Weekly/mo

# take a peek at the data
print(dataset.head())
reviews = np.array(dataset['review'])
sentiments = np.array(dataset['sentiment'])

# build train and test datasets
train_reviews = reviews[:5000]
train_sentiments = sentiments[:5000]
test_reviews = reviews[5000:7000]
test_sentiments = sentiments[5000:7000]

# normalize datasets
#norm_train_reviews = tn.normalize_corpus(train_reviews)
norm_train_reviews = train_reviews
#norm_test_reviews = tn.normalize_corpus(test_reviews)
norm_test_reviews = test_reviews
```

```
                                              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 [10]:

```
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 [11]:

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

In [12]:

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

```
BOW model:> Train features shape: (5000, 434563)  Test features shape: (2000,
434563)
TFIDF model:> Train features shape: (5000, 434563)  Test features shape: (200
```

```
0. 434563)
```

Model Training, Prediction and Performance Evaluation

In [13]:

```
from sklearn.linear_model import SGDClassifier, LogisticRegression

lr = LogisticRegression(penalty='l2', max_iter=1000, C=1)
svm = SGDClassifier(loss='hinge',     max_iter=1000)
```

In [14]:

```
# Logistic Regression model on BOW features
lr_bow_predictions = meu.train_predict_model(classifier=lr,
                                             train_features=cv_train_features
                                             test_features=cv_test_features,

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

```
Model Performance metrics:
------------------------------
Accuracy: 0.8605
Precision: 0.8606
Recall: 0.8605
F1 Score: 0.8605

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

    positive       0.85      0.86      0.86       981
    negative       0.87      0.86      0.86      1019

    accuracy                           0.86      2000
   macro avg       0.86      0.86      0.86      2000
weighted avg       0.86      0.86      0.86      2000


Prediction Confusion Matrix:
------------------------------
                 Predicted:
                   positive negative
Actual: positive        846      135
        negative        144      875
```

In [15]:

```
# Logistic Regression model on TF-IDF features
lr_tfidf_predictions = meu.train_predict_model(classifier=lr,
                                               train_features=tv_train_featur
                                               test_features=tv_test_features
meu.display_model_performance_metrics(true_labels=test_sentiments, predicted_
                                      classes=['positive', 'negative'])
```

```
Model Performance metrics:
------------------------------
Accuracy: 0.866
Precision: 0.8661
Recall: 0.866
```

```
F1 Score: 0.866

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

    positive       0.87      0.85      0.86       981
    negative       0.86      0.88      0.87      1019

    accuracy                           0.87      2000
   macro avg       0.87      0.87      0.87      2000
weighted avg       0.87      0.87      0.87      2000


Prediction Confusion Matrix:
------------------------------
                 Predicted:
                   positive negative
Actual: positive        838      143
```

In [16]:

```
svm_bow_predictions = meu.train_predict_model(classifier=svm,
                                             train_features=cv_train_features
                                             test_features=cv_test_features,
meu.display_model_performance_metrics(true_labels=test_sentiments, predicted_
                                      classes=['positive', 'negative'])
```

```
Model Performance metrics:
------------------------------
Accuracy: 0.8525
Precision: 0.8525
Recall: 0.8525
F1 Score: 0.8525

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

    positive       0.85      0.85      0.85       981
    negative       0.85      0.86      0.86      1019

    accuracy                           0.85      2000
   macro avg       0.85      0.85      0.85      2000
weighted avg       0.85      0.85      0.85      2000


Prediction Confusion Matrix:
------------------------------
                 Predicted:
                   positive negative
Actual: positive        829      152
        negative        143      876
```

In [17]:

```
svm_tfidf_predictions = meu.train_predict_model(classifier=svm,
                                                train_features=tv_train_featu
                                                test_features=tv_test_feature
meu.display_model_performance_metrics(true_labels=test_sentiments, predicted_
                                      classes=['positive', 'negative'])
```

```
Model Performance metrics:
------------------------------
Accuracy: 0.881
Precision: 0.881
Recall: 0.881
F1 Score: 0.881

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

    positive       0.88      0.87      0.88       981
    negative       0.88      0.89      0.88      1019

    accuracy                           0.88      2000
   macro avg       0.88      0.88      0.88      2000
weighted avg       0.88      0.88      0.88      2000


Prediction Confusion Matrix:
------------------------------
                 Predicted:
                   positive negative
Actual: positive        857      124
        negative        114      905
```

Newer Supervised Deep Learning Models

In [50]:

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

import spacy
import nltk
from nltk.tokenize.toktok import ToktokTokenizer

tokenizer = ToktokTokenizer()

nlp = spacy.load('en_core_web_sm')
```

Prediction class label encoding

In [23]:

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

In [24]:

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

```
Sentiment class label map: {'negative': 0, 'positive': 1}
Sample test label transformation:
-----------------------------------
Actual Labels: ['negative' 'negative' 'negative']
Encoded Labels: [0 0 0]
One hot encoded Labels:
 [[1. 0.]
 [1. 0.]
 [1. 0.]]
```

Feature Engineering with word embeddings

In [26]:

```
# build word2vec model
w2v_num_features = 500
w2v_model = gensim.models.Word2Vec(tokenized_train, vector_size=w2v_num_featu
                                   min_count=10, sample=1e-3)
```

In [54]:
```
def averaged_word2vec_vectorizer(corpus, model, num_features):
    vocabulary = set(model.wv.index_to_key)

    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.wv[word])
        if nwords:
            feature_vector = np.divide(feature_vector, nwords)

        return feature_vector

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

In [56]:
```
# generate averaged word vector features from word2vec model
avg_wv_train_features = averaged_word2vec_vectorizer(corpus=tokenized_train,
                                                     num_features=500)
avg_wv_test_features = averaged_word2vec_vectorizer(corpus=tokenized_test, mo
                                                    num_features=500)
```

In [57]:
```
# feature engineering with GloVe model
train_nlp = [nlp(item) for item in norm_train_reviews]
train_glove_features = np.array([item.vector for item in train_nlp])

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

In [58]:
```
print('Word2Vec model:> Train features shape:', avg_wv_train_features.shape,
print('GloVe model:> Train features shape:', train_glove_features.shape, ' Te
```

```
Word2Vec model:> Train features shape: (5000, 500)  Test features shape: (200
0, 500)
GloVe model:> Train features shape: (5000, 96)  Test features shape: (2000, 9
6)
```

Modeling with deep neural networks

Building Deep neural network architecture

In [59]:

```
def construct_deepnn_architecture(num_input_features):
    dnn_model = Sequential()
    dnn_model.add(Dense(512, activation='relu', input_shape=(num_input_featur
    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 [60]:

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

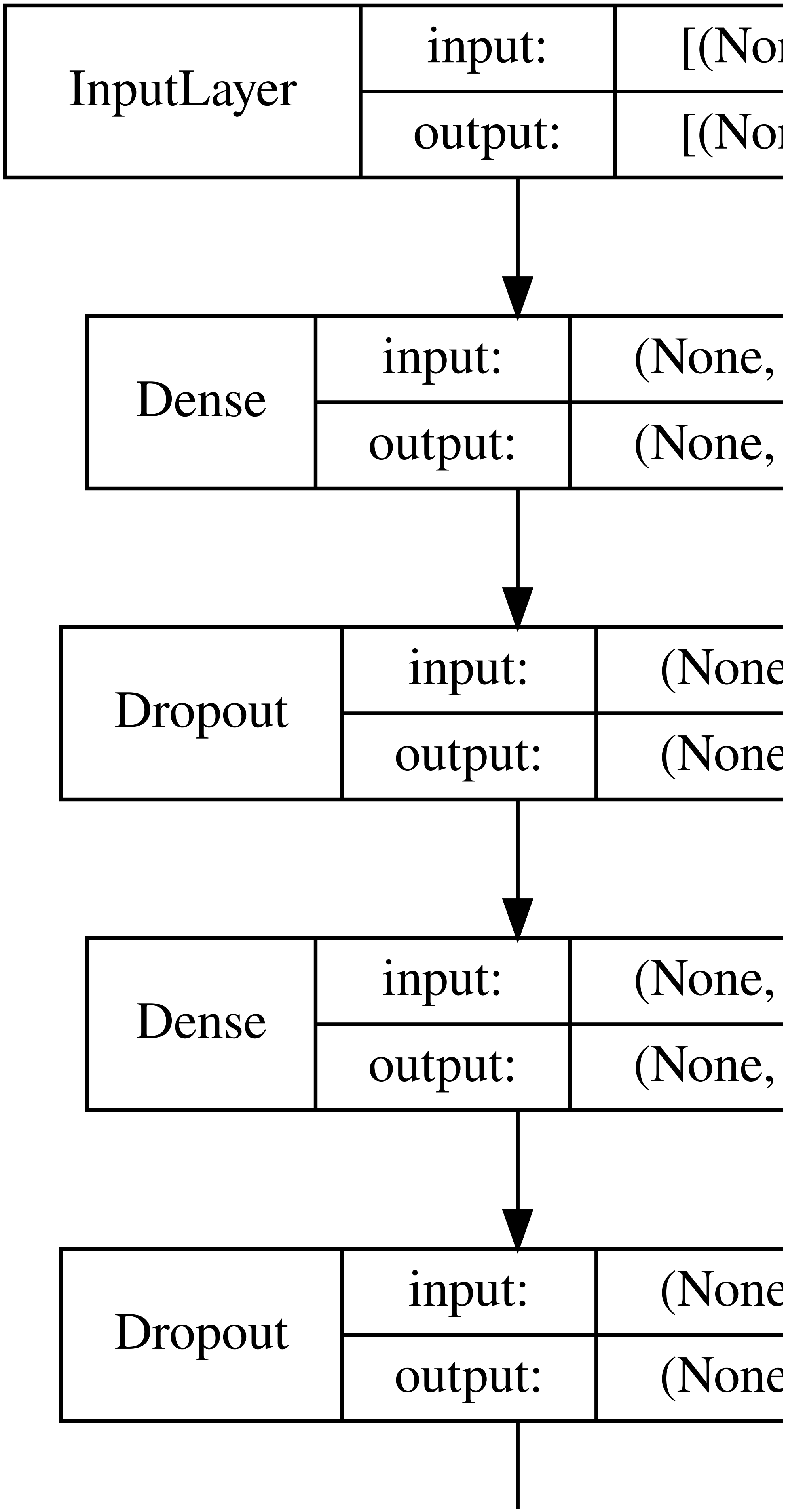
Visualize sample deep architecture

In [61]:

```
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'))
```

Out[61]:


InputLayer
input:
[(Noı
output:
[(Noı
Dense
input:
(None,
output:
(None,
Dropout
input:
(None
output:
(None
Dense
input:
(None,
output:
(None,
Dropout
input:
(None
output:
(None

Model Training, Prediction and Performance Evaluation

In [62]:

```
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)
```

```
Epoch 1/5
45/45 [==============================] - 1s 9ms/step - loss: 0.5115 - accurac
y: 0.7516 - val_loss: 0.4990 - val_accuracy: 0.7640
Epoch 2/5
45/45 [==============================] - 0s 5ms/step - loss: 0.4584 - accurac
y: 0.7884 - val_loss: 0.4626 - val_accuracy: 0.7840
Epoch 3/5
45/45 [==============================] - 0s 4ms/step - loss: 0.4469 - accurac
y: 0.7913 - val_loss: 0.4565 - val_accuracy: 0.7840
Epoch 4/5
45/45 [==============================] - 0s 4ms/step - loss: 0.4452 - accurac
y: 0.7920 - val_loss: 0.4399 - val_accuracy: 0.7820
Epoch 5/5
45/45 [==============================] - 0s 4ms/step - loss: 0.4356 - accurac
y: 0.7938 - val_loss: 0.4483 - val_accuracy: 0.7820
```

Out[62]:

```
<keras.callbacks.History at 0x7f529bcc7e10>
```

In [63]:

```
#y_pred = w2v_dnn.predict_classes(avg_wv_test_features)
y_pred = w2v_dnn.predict(avg_wv_test_features)
y_classes = np.argmax(y_pred,axis=1)
predictions = le.inverse_transform(y_classes)
```

In [64]:

```
import pkg_resources
pkg_resources.get_distribution('gensim').version
```

Out[64]:

```
'4.1.2'
```

In [65]:
```
meu.display_model_performance_metrics(true_labels=test_sentiments, predicted_
                                      classes=['positive', 'negative'])
```

```
Model Performance metrics:
------------------------------
Accuracy: 0.799
Precision: 0.8019
Recall: 0.799
F1 Score: 0.7988

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

    positive       0.77      0.84      0.80       981
    negative       0.83      0.76      0.79      1019

    accuracy                           0.80      2000
   macro avg       0.80      0.80      0.80      2000
weighted avg       0.80      0.80      0.80      2000


Prediction Confusion Matrix:
------------------------------
                 Predicted:
                   positive negative
Actual: positive        826      155
        negative        247      772
```

In [66]:
```
glove_dnn = construct_deepnn_architecture(num_input_features=96)
```

In [67]:
```
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)
```

```
Epoch 1/5
45/45 [==============================] - 1s 7ms/step - loss: 0.6740 - accurac
y: 0.5836 - val_loss: 0.6558 - val_accuracy: 0.6140
Epoch 2/5
45/45 [==============================] - 0s 5ms/step - loss: 0.6528 - accurac
y: 0.6167 - val_loss: 0.6437 - val_accuracy: 0.6180
Epoch 3/5
45/45 [==============================] - 0s 6ms/step - loss: 0.6410 - accurac
y: 0.6293 - val_loss: 0.6375 - val_accuracy: 0.6460
Epoch 4/5
45/45 [==============================] - 0s 6ms/step - loss: 0.6300 - accurac
y: 0.6569 - val_loss: 0.6730 - val_accuracy: 0.5980
Epoch 5/5
45/45 [==============================] - 0s 5ms/step - loss: 0.6393 - accurac
y: 0.6356 - val_loss: 0.6387 - val_accuracy: 0.6300
```

Out[67]:
```
<keras.callbacks.History at 0x7f52ab69e710>
```

In [68]:

```
#y_pred = glove_dnn.predict_classes(test_glove_features)
y_pred = glove_dnn.predict(test_glove_features)
y_classes = np.argmax(y_pred,axis=1)
predictions = le.inverse_transform(y_classes)
```

In [69]:

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

```
Model Performance metrics:
------------------------------
Accuracy: 0.6265
Precision: 0.6488
Recall: 0.6265
F1 Score: 0.6149

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

    positive       0.59      0.81      0.68       981
    negative       0.71      0.45      0.55      1019

    accuracy                           0.63      2000
   macro avg       0.65      0.63      0.62      2000
weighted avg       0.65      0.63      0.61      2000


Prediction Confusion Matrix:
------------------------------
                 Predicted:
                   positive negative
Actual: positive        791      190
        negative        557      462
```