Text Data - Natural Language Processing (NLP)

Text data usually consists of a collection of documents (called the corpus) which can represent words, sentences, or even paragraphs of free flowing text.

The inherent unstructured (no neatly formatted data columns!) and noisy nature of textual data makes it harder for machine learning methods to directly work on raw text data.

Feature Engineering

Feature engineering dramatically improve performance of machine learning models and wins Kaggle competitions. This is especially true for text data, which is unstructured, noisy, and complex.

This section will cover the following types of features for text data

1. Bag of Words
2. Bag of N-Grams (uni-gram, bi-gram, tri-gram, etc.)
3. TF-IDF (term frequency over inverse document frequency)

In [1]:

```
import pandas as pd
import numpy as np
import re
import nltk
import matplotlib.pyplot as plt
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /home/magni/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[1]:

```
True
```

In [2]:

```
corpus = ['The sky is blue and beautiful.',
          'Love this blue and beautiful sky!',
          'The quick brown fox jumps over the lazy dog.',
          'The brown fox is quick and the blue dog is lazy!',
          'The sky is very blue and the sky is very beautiful today',
          'The dog is lazy but the brown fox is quick!'
]

labels = ['weather', 'weather', 'animals', 'animals', 'weather', 'animals']
corpus = np.array(corpus)
corpus_df = pd.DataFrame({'Document': corpus,
                          'Category': labels})
corpus_df = corpus_df[['Document', 'Category']]
corpus_df
```

Out[2]:

Document	Category

	Document	Category
0	The sky is blue and beautiful.	weather
1	Love this blue and beautiful sky!	weather
2	The quick brown fox jumps over the lazy dog.	animals
3	The brown fox is quick and the blue dog is lazy!	animals
4	The sky is very blue and the sky is very beaut...	weather

In [3]:

```
display(type(corpus))
display(corpus)
```

```
numpy.ndarray
array(['The sky is blue and beautiful.',
       'Love this blue and beautiful sky!',
       'The quick brown fox jumps over the lazy dog.',
       'The brown fox is quick and the blue dog is lazy!',
       'The sky is very blue and the sky is very beautiful today',
       'The dog is lazy but the brown fox is quick!'], dtype='<U56')
```

In [4]:

```
corpus_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Document  6 non-null      object
 1   Category  6 non-null      object
dtypes: object(2)
memory usage: 224.0+ bytes
```

Text pre-processing

Depending on your downstream task, cleaning and pre-processing text can involve several different components. Here are a few important components of Natural Language Processing (NLP) pipelines.

1. Removing tags: unnecessary content like HTML tags
2. Removing accented characters: other languages such as French, convert ASCII
3. Removing special characters: adds noise to text, use simple regular expressions (regexes)
4. Stemming and lemmatization: Stemming remove prefixes and suffixes of word stems (i.e. root words), ex. WATCH is the root stem of WATCHES, WATCHING, and WATCHE. Lemmatization similar but lexicographically correct word (present in the dictionary).
5. Expanding contractions: helps text standardization, ex. do not to don't and I would to I'd
6. Removing stopwords: Words without meaningful significance (ex. a, an, the, and) but high frequency.

Additional pre-processing: tokenization, removing extra whitespaces, lower casing and more advanced operations like spelling corrections, grammatical error corrections, removing repeated characters.

In [5]:
```
wpt = nltk.WordPunctTokenizer()
stop_words = nltk.corpus.stopwords.words('english')

def normalize_document(doc):
    # lower case and remove special characters\whitespaces
    doc = re.sub(r'[^a-zA-Z0-9\s]', '', doc, re.I)
    doc = doc.lower()
    doc = doc.strip()
    # tokenize document
    tokens = wpt.tokenize(doc)
    # filter stopwords out of document
    filtered_tokens = [token for token in tokens if token not in stop_words]
    # re-create document from filtered tokens
    doc = ' '.join(filtered_tokens)
    return doc

normalize_corpus = np.vectorize(normalize_document)
```

In [6]:
```
norm_corpus = normalize_corpus(corpus)
norm_corpus
```

Out[6]:
```
array(['sky blue beautiful', 'love blue beautiful sky',
       'quick brown fox jumps lazy dog', 'brown fox quick blue dog lazy',
       'sky blue sky beautiful today', 'dog lazy brown fox quick'],
      dtype='<U30')
```

1. Bag of Words Model

This is perhaps the most simple vector space representational model for unstructured text. A vector space model is simply a mathematical model to represent unstructured text (or any other data) as numeric vectors, such that each dimension of the vector is a specific feature\attribute. The bag of words model represents each text document as a numeric vector where each dimension is a specific word from the corpus and the value could be its frequency in the document, occurrence (denoted by 1 or 0) or even weighted values. The model's name is such because each document is represented literally as a 'bag' of its own words, disregarding word orders, sequences and grammar.

In [7]:
```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(min_df=0., max_df=1.)
cv_matrix = cv.fit_transform(norm_corpus)
cv_matrix = cv_matrix.toarray()
cv_matrix
```

Out[7]:
```
array([[1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0],
       [0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0],
       [0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0],
```

```
       [1, 1, 0, 0, 0, 0, 0, 0, 0, 2, 1],
       [0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0]])
```

Thus you can see that our documents have been converted into numeric vectors such that each document is represented by one vector (row) in the above feature matrix. The following code will help represent this in a more easy to understand format.

In [8]:

```
# get all unique words in the corpus
vocab = cv.get_feature_names()
# show document feature vectors
pd.DataFrame(cv_matrix, columns=vocab)
```

```
/home/magni/python_env/ML1010_env2/lib64/python3.7/site-packages/sklearn/util
s/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated;
get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use
get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

Out[8]:

	beautiful	blue	brown	dog	fox	jumps	lazy	love	quick	sky	today
0	1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	1	0	1	0
2	0	0	1	1	1	1	1	0	1	0	0
3	0	1	1	1	1	0	1	0	1	0	0
4	1	1	0	0	0	0	0	0	0	2	1
5	0	0	1	1	1	0	1	0	1	0	0

This should make things more clearer! You can clearly see that each column or dimension in the feature vectors represents a word from the corpus and each row represents one of our documents. The value in any cell, represents the number of times that word (represented by column) occurs in the specific document (represented by row). Hence if a corpus of documents consists of N unique words across all the documents, we would have an N-dimensional vector for each of the documents.

This should make things more clearer! You can clearly see that each column or dimension in the feature vectors represents a word from the corpus and each row represents one of our documents. The value in any cell, represents the number of times that word (represented by column) occurs in the specific document (represented by row). Hence if a corpus of documents consists of N unique words across all the documents, we would have an N-dimensional vector for each of the documents.

2. Bag of N-Grams Model¶

A word is just a single token, often known as a unigram or 1-gram. We already know that the Bag of Words model doesn't consider order of words. But what if we also wanted to take into account phrases or collection of words which occur in a sequence? N-grams help us achieve that. An N-gram is basically a collection of word tokens from a text document such that these tokens are contiguous and occur in a sequence. Bi-grams indicate n-grams of order 2 (two words), Tri-grams indicate n-grams of order 3 (three words), and so on. The Bag of N-Grams model is hence just an extension of the Bag of Words model so we can also leverage N-gram based features. The following example depicts bi-gram based features in each document feature vector.

In [9]:

```
# you can set the n-gram range to 1,2 to get unigrams as well as bigrams
bv = CountVectorizer(ngram_range=(2,2))
bv_matrix = bv.fit_transform(norm_corpus)

bv_matrix = bv_matrix.toarray()
vocab = bv.get_feature_names()
pd.DataFrame(bv_matrix, columns=vocab)
```

```
/home/magni/python_env/ML1010_env2/lib64/python3.7/site-packages/sklearn/util
s/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated;
get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use
get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

Out[9]:

	beautiful sky	beautiful today	blue beautiful	blue dog	blue sky	brown fox	dog lazy	fox jumps	fox quick	jumps lazy	lazy brown	lazy dog	love blue
0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	1	0	1	0	1	0	1	0
3	0	0	0	1	0	1	1	0	1	0	0	0	0
4	0	1	0	0	1	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	1	0	1	0	1	0	0

3. TF-IDF Model

There are some potential problems which might arise with the Bag of Words model when it is used on large corpora. Since the feature vectors are based on absolute term frequencies, there might be some terms which occur frequently across all documents and these may tend to overshadow other terms in the feature set. The TF-IDF model tries to combat this issue by using a scaling or normalizing factor in its computation. TF-IDF stands for Term Frequency-Inverse Document Frequency, which uses a combination of two metrics in its computation, namely: term frequency (tf) and inverse document frequency (idf). This technique was developed for ranking results for queries in search engines and now it is an indispensable model in the world of information retrieval and NLP.

Mathematically, we can define TF-IDF as tfidf = tf x idf, which can be expanded further to be represented as follows.

Here, tfidf(w, D) is the TF-IDF score for word w in document D. The term tf(w, D) represents the term frequency of the word w in document D, which can be obtained from the Bag of Words model. The term idf(w, D) is the inverse document frequency for the term w, which can be computed as the log transform of the total number of documents in the corpus C divided by the document frequency of the word w, which is basically the frequency of documents in the corpus where the word w occurs. There are multiple variants of this model but they all end up giving quite similar results. Let's apply this on our corpus now!

In [10]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

tv = TfidfVectorizer(min_df=0., max_df=1., use_idf=True)
tv_matrix = tv.fit_transform(norm_corpus)
tv_matrix = tv_matrix.toarray()

vocab = tv.get_feature_names()
pd.DataFrame(np.round(tv_matrix, 2), columns=vocab)
```

```
/home/magni/python_env/ML1010_env2/lib64/python3.7/site-packages/sklearn/util
s/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated;
get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use
get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

Out[10]:

	beautiful	blue	brown	dog	fox	jumps	lazy	love	quick	sky	today
0	0.60	0.52	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.60	0.00
1	0.46	0.39	0.00	0.00	0.00	0.00	0.00	0.66	0.00	0.46	0.00
2	0.00	0.00	0.38	0.38	0.38	0.54	0.38	0.00	0.38	0.00	0.00
3	0.00	0.36	0.42	0.42	0.42	0.00	0.42	0.00	0.42	0.00	0.00
4	0.36	0.31	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.72	0.52
5	0.00	0.00	0.45	0.45	0.45	0.00	0.45	0.00	0.45	0.00	0.00

The TF-IDF based feature vectors for each of our text documents show scaled and normalized values as compared to the raw Bag of Words model values. Interested readers who might want to dive into further details of how the internals of this model work can refer to page 181 of Text Analytics with Python (Springer\Apress; Dipanjan Sarkar, 2016).

Configuration

In [ ]:

```
# Parameters
PROJECT_NAME = 'ML1010_Weekly'
ENABLE_COLAB = True

#Root Machine Learning Directory. Projects appear underneath
GOOGLE_DRIVE_MOUNT = '/content/gdrive'
COLAB_ROOT_DIR = GOOGLE_DRIVE_MOUNT + '/MyDrive/Colab Notebooks'
COLAB_INIT_DIR = COLAB_ROOT_DIR + '/utility_files'

LOCAL_ROOT_DIR = '/home/magni/Documents/ML_Projects'
LOCAL_INIT_DIR = LOCAL_ROOT_DIR + '/utility_files'
```

Bootstrap Environment

In [ ]:

```
#add in support for utility file directory and importing
import sys
import os

if ENABLE_COLAB:
  #Need access to drive
  from google.colab import drive
  drive.mount(GOOGLE_DRIVE_MOUNT, force_remount=True)

  #add in utility directory to syspath to import
  INIT_DIR = COLAB_INIT_DIR
  sys.path.append(os.path.abspath(INIT_DIR))

  #Config environment variables
  ROOT_DIR = COLAB_ROOT_DIR

else:
  #add in utility directory to syspath to import
  INIT_DIR = LOCAL_INIT_DIR
  sys.path.append(os.path.abspath(INIT_DIR))

  #Config environment variables
  ROOT_DIR = LOCAL_ROOT_DIR

#Import Utility Support
from jarvis import Jarvis
jarvis = Jarvis(ROOT_DIR, PROJECT_NAME)

import mv_python_utils as mvutils
```

```
Mounted at /content/gdrive
Wha...where am I?
I am awake now.

I have set your current working directory to /content/gdrive/MyDrive/Colab No
tebooks/ML1010_Weekly
```

```
The current time is 19:35
Hello sir. I hope you had dinner.
```

Emotion and Sentiment Analysis

Sentiment analysis is perhaps one of the most popular applications of NLP, with a vast number of tutorials, courses, and applications that focus on analyzing sentiments of diverse datasets ranging from corporate surveys to movie reviews. The key aspect of sentiment analysis is to analyze a body of text for understanding the opinion expressed by it. Typically, we quantify this sentiment with a positive or negative value, called polarity. The overall sentiment is often inferred as positive, neutral or negative from the sign of the polarity score.

Usually, sentiment analysis works best on text that has a subjective context than on text with only an objective context. Objective text usually depicts some normal statements or facts without expressing any emotion, feelings, or mood. Subjective text contains text that is usually expressed by a human having typical moods, emotions, and feelings. Sentiment analysis is widely used, especially as a part of social media analysis for any domain, be it a business, a recent movie, or a product launch, to understand its reception by the people and what they think of it based on their opinions or, you guessed it, sentiment!

Typically, sentiment analysis for text data can be computed on several levels, including on an individual sentence level, paragraph level, or the entire document as a whole. Often, sentiment is computed on the document as a whole or some aggregations are done after computing the sentiment for individual sentences. There are two major approaches to sentiment analysis.

- Supervised machine learning or deep learning approaches
- Unsupervised lexicon-based approaches

For the first approach we typically need pre-labeled data. Hence, we will be focusing on the second approach. For a comprehensive coverage of sentiment analysis, refer to Chapter 7: Analyzing Movie Reviews Sentiment, Practical Machine Learning with Python, Springer\Apress, 2018. In this scenario, we do not have the convenience of a well-labeled training dataset. Hence, we will need to use unsupervised techniques for predicting the sentiment by using knowledgebases, ontologies, databases, and lexicons that have detailed information, specially curated and prepared just for sentiment analysis. A lexicon is a dictionary, vocabulary, or a book of words. In our case, lexicons are special dictionaries or vocabularies that have been created for analyzing sentiments. Most of these lexicons have a list of positive and negative polar words with some score associated with them, and using various techniques like the position of words, surrounding words, context, parts of speech, phrases, and so on, scores are assigned to the text documents for which we want to compute the sentiment. After aggregating these scores, we get the final sentiment.

Various popular lexicons are used for sentiment analysis, including the following.

AFINN lexicon Bing Liu's lexicon MPQA subjectivity lexicon SentiWordNet VADER lexicon TextBlob lexicon This is not an exhaustive list of lexicons that can be leveraged for sentiment analysis, and there are several other lexicons which can be easily obtained from the Internet. Feel free to check out each of these links and explore them. We will be covering two techniques in this section

Some Pre-Processing

Import necessary depencencies

In [ ]:

```
import pandas as pd
import numpy as np
#import model_evaluation_utils as meu

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

In [ ]:

```
!pip install Afinn
```

```
Collecting Afinn
  Downloading afinn-0.1.tar.gz (52 kB)
     |████████████████████████████████| 52 kB 366 kB/s
Building wheels for collected packages: Afinn
  Building wheel for Afinn (setup.py) ... done
  Created wheel for Afinn: filename=afinn-0.1-py3-none-any.whl size=53448 sha256=c3f0ed2f6827bfc678d09a0b9e8313652b56fd542d69d7dc5d640a0f23e220e6
  Stored in directory: /root/.cache/pip/wheels/9d/16/3a/9f0953027434eab5dadf3f33ab3298fa95afa8292fcf7aba75
Successfully built Afinn
Installing collected packages: Afinn
Successfully installed Afinn-0.1
```

Load and normalize data

1. Cleaning Text - strip HTML
2. Removing accented characters
3. Expanding Contractions
4. Removing Special Characters
5. Lemmatizing text¶
6. Removing Stopwords

In [ ]:
```
dataset = pd.read_csv(jarvis.DATA_DIR + '/movie_reviews_cleaned.csv')

reviews = np.array(dataset['review'])
sentiments = np.array(dataset['sentiment'])

# extract data for model evaluation
train_reviews = reviews[:35000]
train_sentiments = sentiments[:35000]

test_reviews = reviews[35000:]
test_sentiments = sentiments[35000:]
sample_review_ids = [7626, 3533, 13010]
```

In [ ]:
```
# SKIP FOR THE STUDENTS BECAUSE INSTRUCTOR HAS PRE_NORMALIZED AND SAVED THE F
# normalize dataset (time consuming using spacey pipeline)
"""
norm_test_reviews = tn.normalize_corpus(test_reviews)
norm_train_reviews = tn.normalize_corpus(train_reviews)
#output back to a csv file again
import csv
with open(r'movie_reviews_cleaned.csv', mode='w') as cleaned_file:
    csv_writer = csv.writer(cleaned_file, delimiter=',', quotechar='"', quoti
    csv_writer.writerow(['review', 'sentiment'])
    for  text, sent in zip(norm_test_reviews, test_sentiments):
        csv_writer.writerow([text, sent])
    for  text, sent in zip(norm_train_reviews, train_sentiments):
        csv_writer.writerow([text, sent])
"""
```

Out[ ]:
```
'\nnorm_test_reviews = tn.normalize_corpus(test_reviews)\nnorm_train_reviews
= tn.normalize_corpus(train_reviews)\n#output back to a csv file again\nimpor
t csv\nwith open(r\'movie_reviews_cleaned.csv\', mode=\'w\') as cleaned_fil
e:\n    csv_writer = csv.writer(cleaned_file, delimiter=\',\', quotechar
=\'"\', quoting=csv.QUOTE_MINIMAL)\n    csv_writer.writerow([\'review\', \'se
ntiment\'])\n    for  text, sent in zip(norm_test_reviews, test_sentiments):\
n        csv_writer.writerow([text, sent])\n    for  text, sent in zip(norm_t
rain_reviews, train_sentiments):\n        csv_writer.writerow([text, sent])\n
'
```

==

Part A. Unsupervised (Lexicon) Sentiment Analysis

==

1. Sentiment Analysis with AFINN

The AFINN lexicon is perhaps one of the simplest and most popular lexicons that can be used extensively for sentiment analysis. Developed and curated by Finn Arup Nielsen, you can find more details on this lexicon in the paper, "A new ANEW: evaluation of a word list for sentiment analysis in microblogs", proceedings of the ESWC 2011 Workshop. The current version of the lexicon is AFINN-en-165. txt and it contains over 3,300+ words with a polarity score associated with each word. You can find this lexicon at the author's official GitHub repository along with previous versions of it, including AFINN-111. The author has also created a nice wrapper library on top of this in Python called afinn, which we will be using for our analysis.

In [ ]:
```
from afinn import Afinn

afn = Afinn(emoticons=True)

# NOTE:  to use afinn score, call the function afn.score("text you want the s
# the lexicon will be used to compute summary of sentiment for the given text
```

Predict sentiment for sample reviews

We can get a good idea of general sentiment for different sample.

In [ ]:
```
for review, sentiment in zip(test_reviews[sample_review_ids], test_sentiments
    print('REVIEW:', review)
    print('Actual Sentiment:', sentiment)
    print('Predicted Sentiment polarity:', afn.score(review))
    print('-'*60)
```

```
REVIEW: word fail whenever want describe feeling movie sequel flaw sure start
subspecie not execute well enough special effect glorify movie herd movie mas
s consumer care quantity quality cheap fun depth crap like blade not even des
erve capital letter underworlddracula 2000dracula 3000 good movie munch popco
rn drink couple coke make subspecie superior effort anyone claim vampire fana
tic hand obvious vampire romanian story set transylvania scene film location
convince atmosphere not base action pack chase expensive orchestral music rad
u source atmosphere vampire look like behave add breathtakingly gloomy castle
dark passageway situate romania include typical vampiric element movement sha
dow wall vampire take flight work art short like fascinated vampire feel appe
arance well setting sinister dark no good place look subspecie movie vampire
journal brilliant spin former
Actual Sentiment: positive
Predicted Sentiment polarity: 20.0
------------------------------------------------------------
REVIEW: good family movie laugh wish not much school stuff like bully fill mo
vie also seem little easy save piece land build mean flow easily make aware w
ildlife cute way introduce piece land fast runner little slow little hokey re
mind go back school oh dvd chock full goody not miss 7 10 movie 10 10 dvd ext
ra well worth watch well worth time see
Actual Sentiment: positive
Predicted Sentiment polarity: 12.0
------------------------------------------------------------
REVIEW: opinion movie not good hardly find good thing say still would like ex
plain conclude another bad movie decide watch costas mandylor star main reaso
n watch till end like action movie understand movie build action rather story
know not go detail come credibility story event even not explain scene lack s
ense reality look ridiculous beginning movie look quite promising tough good
```

```
look specialist not tough smart funny partner must job turn bit different exp
ect story take place cruise ship disaster happen ship turn leave alive strugg
le survive escape shark professional killer rise water furthermore movie quit
e violent main weapon beside disaster already take passenger gun successfully
use many case personally miss good man man woman woman prefer fight family fu
n not think think movie shoot hurry without real vision try say make usual ac
tion movie trick bit something call love without real meaning result bad movi
e
Actual Sentiment: negative
Predicted Sentiment polarity: 2.0
------------------------------------------------------------
```

Predict sentiment for test dataset

In [ ]:

```
sentiment_polarity = [afn.score(review) for review in test_reviews]
predicted_sentiments = ['positive' if score >= 1.0 else 'negative' for score
```

In [ ]:

```
display(type(sentiment_polarity))
print(sentiment_polarity[4])
```

```
list
12.0
```

Evaluate model performance

In [ ]:

```
from sklearn import metrics

results = metrics.classification_report(test_sentiments, predicted_sentiments
print(results)

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

```
              precision    recall  f1-score   support

    negative       0.78      0.56      0.65      7413
    positive       0.66      0.84      0.74      7587

    accuracy                           0.71     15000
   macro avg       0.72      0.70      0.70     15000
weighted avg       0.72      0.71      0.70     15000
```

2. Sentiment Analysis with SentiWordNet

SentiWordNet is a lexical resource for opinion mining. SentiWordNet assigns to each synset of WordNet three sentiment scores: positivity, negativity, objectivity. SentiWordNet is described in details in the papers:

In [ ]:
```
from nltk.corpus import sentiwordnet as swn
import nltk
nltk.download('sentiwordnet')

awesome = list(swn.senti_synsets('awesome', 'a'))[0]
print('Positive Polarity Score:', awesome.pos_score())
print('Negative Polarity Score:', awesome.neg_score())
print('Objective Score:', awesome.obj_score())
```

```
[nltk_data] Downloading package sentiwordnet to
[nltk_data]     /home/anniee/nltk_data...
[nltk_data]   Package sentiwordnet is already up-to-date!
Positive Polarity Score: 0.875
Negative Polarity Score: 0.125
Objective Score: 0.0
```

Build model

For each word in the review, add up the sentiment score of words that are NN, VB, JJ, RB if it's in the lexicon dictionary.

In [ ]:
```
import text_normalizer as tn

def analyze_sentiment_sentiwordnet_lexicon(review,
                                           verbose=False):

    # tokenize and POS tag text tokens
    tagged_text = [(token.text, token.tag_) for token in tn.nlp(review)]
    pos_score = neg_score = token_count = obj_score = 0
    # get wordnet synsets based on POS tags
    # get sentiment scores if synsets are found
    for word, tag in tagged_text:
        ss_set = None
        if 'NN' in tag and list(swn.senti_synsets(word, 'n')):
            ss_set = list(swn.senti_synsets(word, 'n'))[0]
        elif 'VB' in tag and list(swn.senti_synsets(word, 'v')):
            ss_set = list(swn.senti_synsets(word, 'v'))[0]
        elif 'JJ' in tag and list(swn.senti_synsets(word, 'a')):
            ss_set = list(swn.senti_synsets(word, 'a'))[0]
        elif 'RB' in tag and list(swn.senti_synsets(word, 'r')):
            ss_set = list(swn.senti_synsets(word, 'r'))[0]
        # if senti-synset is found
        if ss_set:
            # add scores for all found synsets
            pos_score += ss_set.pos_score()
            neg_score += ss_set.neg_score()
            obj_score += ss_set.obj_score()
            token_count += 1

    # aggregate final scores
    final_score = pos_score - neg_score
    norm_final_score = round(float(final_score) / token_count, 2)
    final_sentiment = 'positive' if norm_final_score >= 0 else 'negative'
    if verbose:
        norm_obj_score = round(float(obj_score) / token_count, 2)
        norm_pos_score = round(float(pos_score) / token_count, 2)
        norm_neg_score = round(float(neg_score) / token_count, 2)
        # to display results in a nice table
        sentiment_frame = pd.DataFrame([[final_sentiment, norm_obj_score, nor
                                         norm_neg_score, norm_final_score]],
                                       columns=pd.MultiIndex(levels=[['SENTIM
                                                             ['Predicted Sent
                                                              'Positive', 'Ne
                                                             labels=[[0,0,0,0
        print(sentiment_frame)

    return final_sentiment
```

Predict sentiment for sample reviews

In [ ]:
```
for review, sentiment in zip(test_reviews[sample_review_ids], test_sentiments
    print('REVIEW:', review)
    print('Actual Sentiment:', sentiment)
    pred = analyze_sentiment_sentiwordnet_lexicon(review, verbose=True)
    print('-'*60)
```

```
REVIEW: word fail whenever want describe feeling movie sequel flaw sure start
```

```
subspecie not execute well enough special effect glorify movie herd movie mas
s consumer care quantity quality cheap fun depth crap like blade not even des
erve capital letter underworlddracula 2000dracula 3000 good movie munch popco
rn drink couple coke make subspecie superior effort anyone claim vampire fana
tic hand obvious vampire romanian story set transylvania scene film location 
convince atmosphere not base action pack chase expensive orchestral music rad
u source atmosphere vampire look like behave add breathtakingly gloomy castle
dark passageway situate romania include typical vampiric element movement sha
dow wall vampire take flight work art short like fascinated vampire feel appe
arance well setting sinister dark no good place look subspecie movie vampire 
journal brilliant spin former
Actual Sentiment: positive
     SENTIMENT STATS:
  Predicted Sentiment Objectivity Positive Negative Overall
0            positive        0.84     0.09     0.06    0.03
------------------------------------------------------------
REVIEW: good family movie laugh wish not much school stuff like bully fill mo
vie also seem little easy save piece land build mean flow easily make aware w
ildlife cute way introduce piece land fast runner little slow little hokey re
mind go back school oh dvd chock full goody not miss 7 10 movie 10 10 dvd ext
ra well worth watch well worth time see
Actual Sentiment: positive
     SENTIMENT STATS:
  Predicted Sentiment Objectivity Positive Negative Overall
0            positive        0.85     0.08     0.06    0.02
------------------------------------------------------------
REVIEW: opinion movie not good hardly find good thing say still would like ex
plain conclude another bad movie decide watch costas mandylor star main reaso
n watch till end like action movie understand movie build action rather story
know not go detail come credibility story event even not explain scene lack s
ense reality look ridiculous beginning movie look quite promising tough good 
look specialist not tough smart funny partner must job turn bit different exp
ect story take place cruise ship disaster happen ship turn leave alive strugg
le survive escape shark professional killer rise water furthermore movie quit
e violent main weapon beside disaster already take passenger gun successfully
use many case personally miss good man man woman woman prefer fight family fu
n not think think movie shoot hurry without real vision try say make usual ac
tion movie trick bit something call love without real meaning result bad movi
e
Actual Sentiment: negative
     SENTIMENT STATS:
  Predicted Sentiment Objectivity Positive Negative Overall
0            positive        0.82     0.09     0.09    -0.0
------------------------------------------------------------
```

Predict sentiment for test dataset

In [ ]:

```python
predicted_sentiments = [analyze_sentiment_sentiwordnet_lexicon(review, verbos
```

Evaluate model performance

In [ ]:

```python
results = metrics.classification_report(test_sentiments, predicted_sentiments
print(results)
```

```
             precision    recall  f1-score   support
```

```
    negative       0.71      0.60      0.65      7413
    positive       0.66      0.76      0.71      7587

   micro avg       0.68      0.68      0.68     15000
   macro avg       0.69      0.68      0.68     15000
weighted avg       0.69      0.68      0.68     15000
```

3. Sentiment Analysis with VADER

In [ ]:

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

```
/home/anniee/.local/lib/python3.6/site-packages/nltk/twitter/__init__.py:20:
UserWarning: The twython library has not been installed. Some functionality f
rom the twitter package will not be available.
  warnings.warn("The twython library has not been installed. "
```

Build model

In [ ]:

```
def analyze_sentiment_vader_lexicon(review,
                                    threshold=0.1,
                                    verbose=False):
    # pre-process text
    review = tn.strip_html_tags(review)
    review = tn.remove_accented_chars(review)
    review = tn.expand_contractions(review)

    # analyze the sentiment for review
    analyzer = SentimentIntensityAnalyzer()
    scores = analyzer.polarity_scores(review)
    # get aggregate scores and final sentiment
    agg_score = scores['compound']
    final_sentiment = 'positive' if agg_score >= threshold\
                                   else 'negative'
    if verbose:
        # display detailed sentiment statistics
        positive = str(round(scores['pos'], 2)*100)+'%'
        final = round(agg_score, 2)
        negative = str(round(scores['neg'], 2)*100)+'%'
        neutral = str(round(scores['neu'], 2)*100)+'%'
        sentiment_frame = pd.DataFrame([[final_sentiment, final, positive,
                                        negative, neutral]],
                                        columns=pd.MultiIndex(levels=[['SENTI
                                                                      ['Predi
                                                                       'Posit
                                                              labels=[[0,0,0,
        print(sentiment_frame)

    return final_sentiment
```

Predict sentiment for sample reviews

In [ ]:
```
nltk.download('vader_lexicon')

for review, sentiment in zip(test_reviews[sample_review_ids], test_sentiments
    print('REVIEW:', review)
    print('Actual Sentiment:', sentiment)
    pred = analyze_sentiment_vader_lexicon(review, threshold=0.4, verbose=Tru
    print('-'*60)
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /home/anniee/nltk_data...
REVIEW: word fail whenever want describe feeling movie sequel flaw sure start
subspecie not execute well enough special effect glorify movie herd movie mas
s consumer care quantity quality cheap fun depth crap like blade not even des
erve capital letter underworlddracula 2000dracula 3000 good movie munch popco
rn drink couple coke make subspecie superior effort anyone claim vampire fana
tic hand obvious vampire romanian story set transylvania scene film location
convince atmosphere not base action pack chase expensive orchestral music rad
u source atmosphere vampire look like behave add breathtakingly gloomy castle
dark passageway situate romania include typical vampiric element movement sha
dow wall vampire take flight work art short like fascinated vampire feel appe
arance well setting sinister dark no good place look subspecie movie vampire
journal brilliant spin former
Actual Sentiment: positive
     SENTIMENT STATS:
  Predicted Sentiment Polarity Score             Positive Negative Neutral
0            positive           0.98  28.000000000000004%    11.0%   61.0%
------------------------------------------------------------
REVIEW: good family movie laugh wish not much school stuff like bully fill mo
vie also seem little easy save piece land build mean flow easily make aware w
ildlife cute way introduce piece land fast runner little slow little hokey re
mind go back school oh dvd chock full goody not miss 7 10 movie 10 10 dvd ext
ra well worth watch well worth time see
Actual Sentiment: positive
     SENTIMENT STATS:
  Predicted Sentiment Polarity Score Positive Negative             Neutral
0            positive           0.97    39.0%     4.0%  57.99999999999999%
------------------------------------------------------------
REVIEW: opinion movie not good hardly find good thing say still would like ex
plain conclude another bad movie decide watch costas mandylor star main reaso
n watch till end like action movie understand movie build action rather story
know not go detail come credibility story event even not explain scene lack s
ense reality look ridiculous beginning movie look quite promising tough good
look specialist not tough smart funny partner must job turn bit different exp
ect story take place cruise ship disaster happen ship turn leave alive strugg
le survive escape shark professional killer rise water furthermore movie quit
e violent main weapon beside disaster already take passenger gun successfully
use many case personally miss good man man woman woman prefer fight family fu
n not think think movie shoot hurry without real vision try say make usual ac
tion movie trick bit something call love without real meaning result bad movi
e
Actual Sentiment: negative
     SENTIMENT STATS:
  Predicted Sentiment Polarity Score Positive Negative             Neutral
0            negative          -0.98    12.0%    31.0%  56.00000000000001%
------------------------------------------------------------
```

Predict sentiment for test dataset

In [ ]:
```
predicted_sentiments = [analyze_sentiment_vader_lexicon(review, threshold=0.4
```

Evaluate model performance

In [ ]:
```
display_model_performance_metrics(true_labels=test_sentiments, predicted_labe
                                  classes=['positive', 'negative'])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-5c81cb959a93> in <module>()
----> 1 display_model_performance_metrics(true_labels=test_sentiments, predic
ted_labels=predicted_sentiments,
      2                                   classes=['positive', 'negative'])

NameError: name 'display_model_performance_metrics' is not defined
```

In [ ]:

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
```

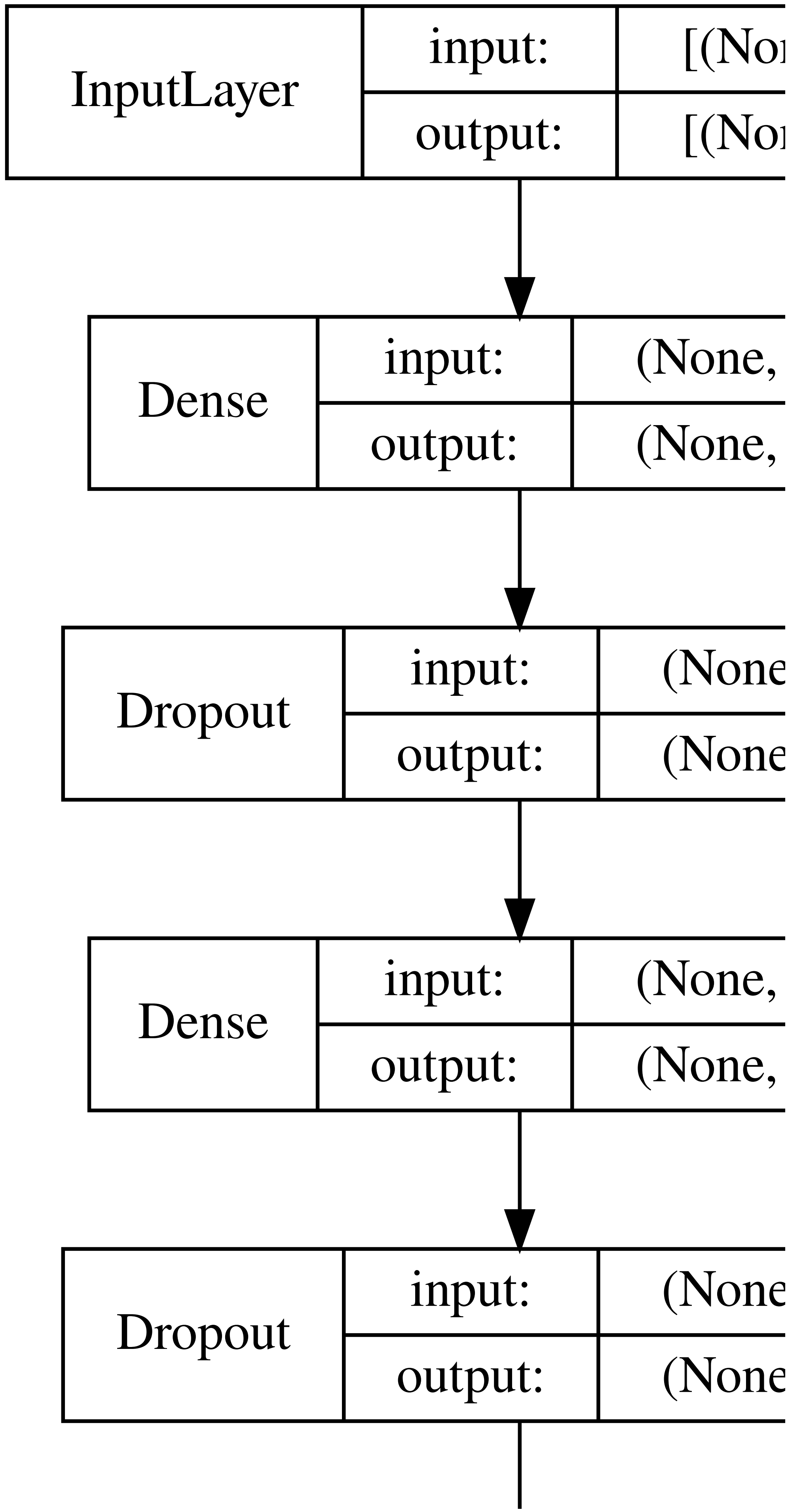
In [60]:
```
w2v_dnn = construct_deepnn_architecture(num_input_features=500)
```

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:
[(Nor
output:
[(Nor
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
```