How to Develop a Deep Learning Bag-of-Words Model for Sentiment Analysis (Text Classification)

by **Jason Brownlee** on September 3, 2020 in **Deep Learning for Natural Language Processing** 129

Movie reviews can be classified as either favorable or not.

The evaluation of movie review text is a classification problem often called sentiment analysis. A popular technique for developing sentiment analysis models is to use a bag-of-words model that transforms documents into vectors where each word in the document is assigned a score.

In this tutorial, you will discover how you can develop a deep learning predictive model using the bag-of-words representation for movie review sentiment classification.

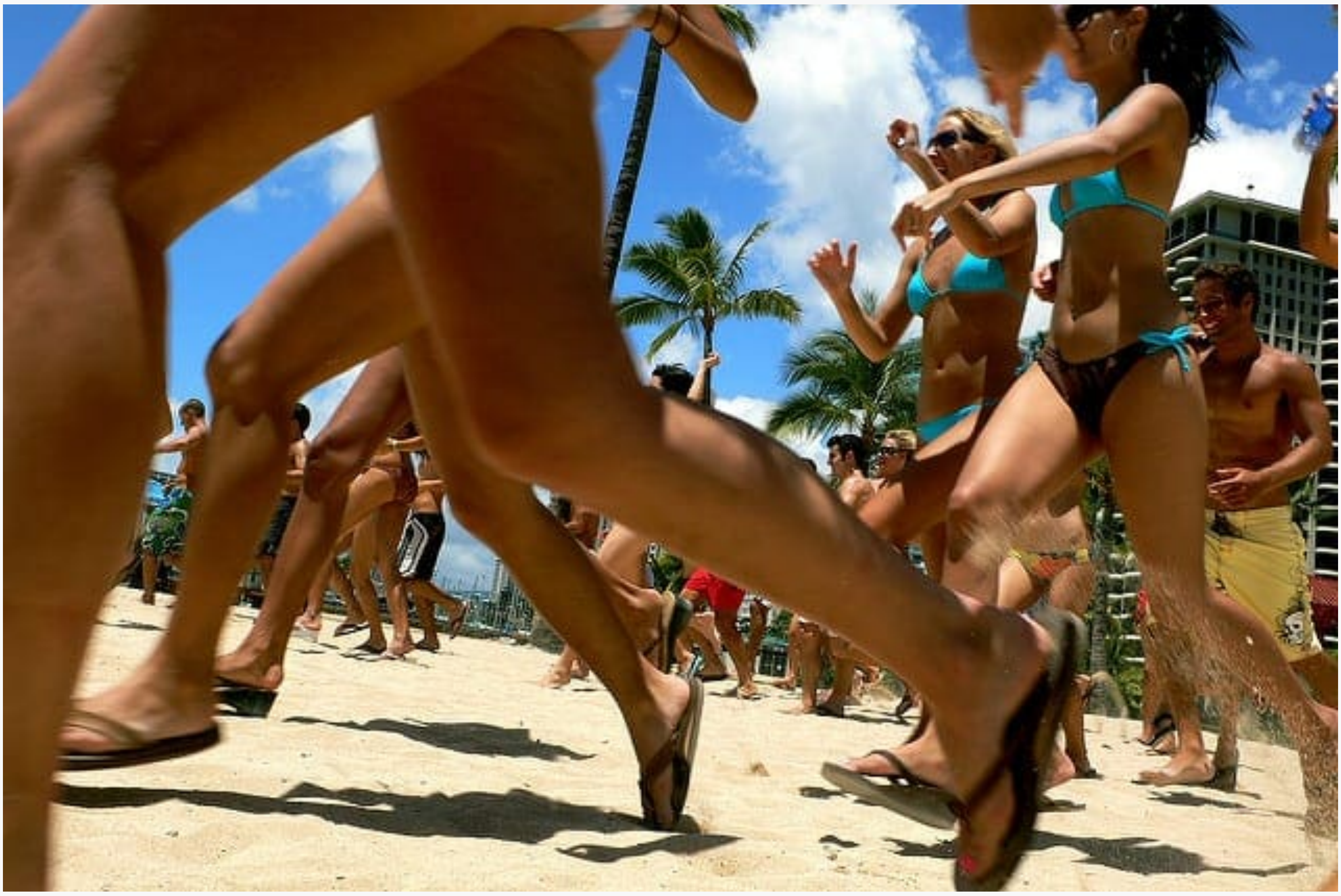
After completing this tutorial, you will know:

- How to prepare the review text data for modeling with a restricted vocabulary.
- How to use the bag-of-words model to prepare train and test data.
- How to develop a multilayer Perceptron bag-of-words model and use it to make predictions on new review text data.

Kick-start your project with my new book Deep Learning for Natural Language Processing, including *step-by-step tutorials* and the *Python source code* files for all examples.

Let’s get started.

- **Update Oct/2017**: Fixed a minor typo when loading and naming positive and negative reviews (thanks Arthur).
- **Update Aug/2020**: Updated link to movie review dataset.



How to Develop a Deep Learning Bag-of-Words Model for Predicting Sentiment in Movie Reviews

Photo by jai Mansson, some rights reserved.

Tutorial Overview

This tutorial is divided into 4 parts; they are:

1. Movie Review Dataset
2. Data Preparation
3. Bag-of-Words Representation
4. Sentiment Analysis Models

Movie Review Dataset

The Movie Review Data is a collection of movie reviews retrieved from the imdb.com website in the early 2000s by Bo Pang and Lillian Lee. The reviews were collected and made available as part of their research on natural language processing.

The reviews were originally released in 2002, but an updated and cleaned up version were released in 2004, referred to as “v2.0”.

The dataset is comprised of 1,000 positive and 1,000 negative movie reviews drawn from an archive of the rec.arts.movies.reviews newsgroup hosted at imdb.com. The authors refer to this dataset as the “polarity dataset”.

> *Our data contains 1000 positive and 1000 negative reviews all written before 2002, with a cap of 20 reviews per author (312 authors total) per category. We refer to this corpus as the polarity dataset.*

— A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts, 2004.

The data has been cleaned up somewhat, for example:

- The dataset is comprised of only English reviews.
- All text has been converted to lowercase.
- There is white space around punctuation like periods, commas, and brackets.
- Text has been split into one sentence per line.

The data has been used for a few related natural language processing tasks. For classification, the performance of classical models (such as Support Vector Machines) on the data is in the range of high 70% to low 80% (e.g. 78%-82%).

More sophisticated data preparation may see results as high as 86% with 10-fold cross validation. This gives us a ballpark of low-to-mid 80s if we were looking to use this dataset in experiments on modern methods.

> *… depending on choice of downstream polarity classifier, we can achieve highly statistically significant improvement (from 82.8% to 86.4%)*

— A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts, 2004.

You can download the dataset from here:

- Movie Review Polarity Dataset (review_polarity.tar.gz, 3MB)

After unzipping the file, you will have a directory called "txt_sentoken" with two sub-directories containing the text "*neg*" and "*pos*" for negative and positive reviews. Reviews are stored one per file with a naming convention *cv000* to *cv999* for each neg and pos.

Next, let's look at loading and preparing the text data.

Data Preparation

In this section, we will look at 3 things:

1. Separation of data into training and test sets.
2. Loading and cleaning the data to remove punctuation and numbers.
3. Defining a vocabulary of preferred words.

Split into Train and Test Sets

We are pretending that we are developing a system that can predict the sentiment of a textual movie review as either positive or negative.

This means that after the model is developed, we will need to make predictions on new textual reviews. This will require all of the same data preparation to be performed on those new reviews as is performed on the training data for the model.

We will ensure that this constraint is built into the evaluation of our models by splitting the training and test datasets prior to any data preparation. This means that any knowledge in the test set that could help us better prepare the data (e.g. the words used) is unavailable during the preparation of data and the training of the model.

That being said, we will use the last 100 positive reviews and the last 100 negative reviews as a test set (100 reviews) and the remaining 1,800 reviews as the training dataset.

This is a 90% train, 10% split of the data.

The split can be imposed easily by using the filenames of the reviews where reviews named 000 to 899 are for training data and reviews named 900 onwards are for testing the model.

Loading and Cleaning Reviews

The text data is already pretty clean, so not much preparation is required.

Without getting too much into the details, we will prepare the data using the following method:

- Split tokens on white space.
- Remove all punctuation from words.
- Remove all words that are not purely comprised of alphabetical characters.
- Remove all words that are known stop words.
- Remove all words that have a length <= 1 character.

We can put all of these steps into a function called clean_doc() that takes as an argument the raw text loaded from a file and returns a list of cleaned tokens. We can also define a function load_doc() that loads a document from file ready for use with the clean_doc() function.

An example of cleaning the first positive review is listed below.

```
from nltk.corpus import stopwords
import string

# load doc into memory
def load_doc(filename):
 # open the file as read only
 file = open(filename, 'r')
 # read all text
 text = file.read()
 # close the file
 file.close()
 return text

# turn a doc into clean tokens
def clean_doc(doc):
 # split into tokens by white space
 tokens = doc.split()
 # remove punctuation from each token
 table = str.maketrans('', '', string.punctuation)
 tokens = [w.translate(table) for w in tokens]
 # remove remaining tokens that are not alphabetic
 tokens = [word for word in tokens if word.isalpha()]
```

```
	# filter out stop words
	stop_words = set(stopwords.words('english'))
	tokens = [w for w in tokens if not w in stop_words]
	# filter out short tokens
	tokens = [word for word in tokens if len(word) > 1]
	return tokens

# load the document
filename = 'txt_sentoken/pos/cv000_29590.txt'
text = load_doc(filename)
tokens = clean_doc(text)
print(tokens)
```

Running the example prints a long list of clean tokens.

There are many more cleaning steps we may want to explore, and I leave them as further exercises. I'd love to see what you can come up with.

```
...
'creepy', 'place', 'even', 'acting', 'hell', 'solid', '
```

Define a Vocabulary

It is important to define a vocabulary of known words when using a bag-of-words model.

The more words, the larger the representation of documents, therefore it is important to constrain the words to only those believed to be predictive. This is difficult to know beforehand and often it is important to test different hypotheses about how to construct a useful vocabulary.

We have already seen how we can remove punctuation and numbers from the vocabulary in the previous section. We can repeat this for all documents and build a set of all known words.

We can develop a vocabulary as a *Counter*, which is a dictionary mapping of words and their count that allows us to easily update and query.

Each document can be added to the counter (a new function called *add_doc_to_vocab()*) and we can step over all of the reviews in the negative directory and then the positive directory (a new function called *process_docs()*).

The complete example is listed below.

```
from string import punctuation
from os import listdir
from collections import Counter
from nltk.corpus import stopwords

# load doc into memory
def load_doc(filename):
 # open the file as read only
 file = open(filename, 'r')
 # read all text
 text = file.read()
 # close the file
 file.close()
 return text

# turn a doc into clean tokens
def clean_doc(doc):
 # split into tokens by white space
 tokens = doc.split()
 # remove punctuation from each token
 table = str.maketrans('', '', punctuation)
 tokens = [w.translate(table) for w in tokens]
 # remove remaining tokens that are not alphabetic
 tokens = [word for word in tokens if word.isalpha()]
 # filter out stop words
 stop_words = set(stopwords.words('english'))
 tokens = [w for w in tokens if not w in stop_words]
 # filter out short tokens
```

```
 tokens = [word for word in tokens if len(word) > 1]
 return tokens

# load doc and add to vocab
def add_doc_to_vocab(filename, vocab):
 # load doc
 doc = load_doc(filename)
 # clean doc
 tokens = clean_doc(doc)
 # update counts
 vocab.update(tokens)

# load all docs in a directory
def process_docs(directory, vocab):
 # walk through all files in the folder
 for filename in listdir(directory):
 # skip any reviews in the test set
 if filename.startswith('cv9'):
 continue
 # create the full path of the file to open
 path = directory + '/' + filename
 # add doc to vocab
 add_doc_to_vocab(path, vocab)

# define vocab
vocab = Counter()
# add all docs to vocab
process_docs('txt_sentoken/pos', vocab)
process_docs('txt_sentoken/neg', vocab)
# print the size of the vocab
print(len(vocab))
# print the top words in the vocab
print(vocab.most_common(50))
```

Running the example shows that we have a vocabulary of 44,276 words.

We also can see a sample of the top 50 most used words in the movie reviews.

Note that this vocabulary was constructed based on only those reviews in the training dataset.

```
44276
[('film', 7983), ('one', 4946), ('movie', 4826), ('like
```

We can step through the vocabulary and remove all words that have a low occurrence, such as only being used once or twice in all reviews.

For example, the following snippet will retrieve only the tokens that appear 2 or more times in all reviews.

```
# keep tokens with a min occurrence
min_occurane = 2
tokens = [k for k,c in vocab.items() if c >= min_occura
print(len(tokens))
```

Running the above example with this addition shows that the vocabulary size drops by a little more than half its size, from 44,276 to 25,767 words.

```
25767
```

Finally, the vocabulary can be saved to a new file called vocab.txt that we can later load and use to filter movie reviews prior to encoding them for modeling. We define a new function called save_list() that saves the vocabulary to file, with one word per file.

For example:

```
# save list to file
def save_list(lines, filename):
 # convert lines to a single blob of text
 data = '\n'.join(lines)
 # open file
 file = open(filename, 'w')
 # write text
 file.write(data)
```

```
# close file
file.close()

# save tokens to a vocabulary file
save_list(tokens, 'vocab.txt')
```

Running the min occurrence filter on the vocabulary and saving it to file, you should now have a new file called *vocab.txt* with only the words we are interested in.

The order of words in your file will differ, but should look something like the following:

```
aberdeen
dupe
burt
libido
hamlet
arlene
available
corners
web
columbia
...
```

We are now ready to look at extracting features from the reviews ready for modeling.

Bag-of-Words Representation

In this section, we will look at how we can convert each review into a representation that we can provide to a Multilayer Perceptron model.

A bag-of-words model is a way of extracting features from text so the text input can be used with machine learning algorithms like neural networks.

Each document, in this case a review, is converted into a vector representation. The number of items in the vector representing a document corresponds to the number of words in the vocabulary. The larger the vocabulary, the longer the vector representation, hence the preference for smaller vocabularies in the previous section.

Words in a document are scored and the scores are placed in the corresponding location in the representation. We will look at different word scoring methods in the next section.

In this section, we are concerned with converting reviews into vectors ready for training a first neural network model.

This section is divided into 2 steps:

1. Converting reviews to lines of tokens.
2. Encoding reviews with a bag-of-words model representation.

Reviews to Lines of Tokens

Before we can convert reviews to vectors for modeling, we must first clean them up.

This involves loading them, performing the cleaning operation developed above, filtering out words not in the chosen vocabulary, and converting the remaining tokens into a single string or line ready for encoding.

First, we need a function to prepare one document. Below lists the function *doc_to_line()* that will load a document, clean it, filter out tokens not in the vocabulary, then return the document as a string of white space separated tokens.

```
# load doc, clean and return line of tokens
def doc_to_line(filename, vocab):
 # load the doc
 doc = load_doc(filename)
 # clean doc
 tokens = clean_doc(doc)
 # filter by vocab
 tokens = [w for w in tokens if w in vocab]
 return ' '.join(tokens)
```

Next, we need a function to work through all documents in a directory (such as '*pos*' and '*neg*') to convert the documents into lines.

Below lists the *process_docs()* function that does just this, expecting a directory name and a vocabulary set as input arguments and returning a list of processed documents.

```
# load all docs in a directory
def process_docs(directory, vocab):
 lines = list()
 # walk through all files in the folder
 for filename in listdir(directory):
 # skip any reviews in the test set
 if filename.startswith('cv9'):
 continue
 # create the full path of the file to open
 path = directory + '/' + filename
 # load and clean the doc
 line = doc_to_line(path, vocab)
 # add to list
 lines.append(line)
 return lines
```

Finally, we need to load the vocabulary and turn it into a set for use in cleaning reviews.

```
# load the vocabulary
vocab_filename = 'vocab.txt'
vocab = load_doc(vocab_filename)
vocab = vocab.split()
vocab = set(vocab)
```

We can put all of this together, reusing the loading and cleaning functions developed in previous sections.

The complete example is listed below, demonstrating how to prepare the positive and negative reviews from the training dataset.

```
from string import punctuation
from os import listdir
from collections import Counter
from nltk.corpus import stopwords

# load doc into memory
def load_doc(filename):
 # open the file as read only
 file = open(filename, 'r')
 # read all text
 text = file.read()
 # close the file
 file.close()
 return text

# turn a doc into clean tokens
def clean_doc(doc):
 # split into tokens by white space
 tokens = doc.split()
 # remove punctuation from each token
 table = str.maketrans('', '', punctuation)
 tokens = [w.translate(table) for w in tokens]
 # remove remaining tokens that are not alphabetic
 tokens = [word for word in tokens if word.isalpha()]
 # filter out stop words
 stop_words = set(stopwords.words('english'))
 tokens = [w for w in tokens if not w in stop_words]
 # filter out short tokens
```

```
    tokens = [word for word in tokens if len(word) > 1]
    return tokens

# load doc, clean and return line of tokens
def doc_to_line(filename, vocab):
    # load the doc
    doc = load_doc(filename)
    # clean doc
    tokens = clean_doc(doc)
    # filter by vocab
    tokens = [w for w in tokens if w in vocab]
    return ' '.join(tokens)

# load all docs in a directory
def process_docs(directory, vocab):
    lines = list()
    # walk through all files in the folder
    for filename in listdir(directory):
        # skip any reviews in the test set
        if filename.startswith('cv9'):
            continue
        # create the full path of the file to open
        path = directory + '/' + filename
        # load and clean the doc
        line = doc_to_line(path, vocab)
        # add to list
        lines.append(line)
    return lines

# load the vocabulary
vocab_filename = 'vocab.txt'
vocab = load_doc(vocab_filename)
vocab = vocab.split()
vocab = set(vocab)
# load all training reviews
positive_lines = process_docs('txt_sentoken/pos', voca
negative_lines = process_docs('txt_sentoken/neg', voca
# summarize what we have
print(len(positive_lines), len(negative_lines))
```

Movie Reviews to Bag-of-Words Vectors

We will use the Keras API to convert reviews to encoded document vectors.

Keras provides the Tokenize class that can do some of the cleaning and vocab definition tasks that we took care of in the previous section.

It is better to do this ourselves to know exactly what was done and why. Nevertheless, the Tokenizer class is convenient and will easily transform documents into encoded vectors.

First, the Tokenizer must be created, then fit on the text documents in the training dataset.

In this case, these are the aggregation of the *positive_lines* and *negative_lines* arrays developed in the previous section.

```
# create the tokenizer
tokenizer = Tokenizer()
# fit the tokenizer on the documents
docs = positive_lines + negative_lines
tokenizer.fit_on_texts(docs)
```

This process determines a consistent way to convert the vocabulary to a fixed-length vector with 25,768 elements, which is the total number of words in the vocabulary file *vocab.txt*.

Next, documents can then be encoded using the Tokenizer by calling *texts_to_matrix()*. The function takes both a list of documents to encode and an encoding mode, which is the method used to score words in the document. Here we specify '*freq*' to score words based on their frequency in the document.

This can be used to encode the training data, for example:

```
# encode training data set
Xtrain = tokenizer.texts_to_matrix(docs, mode='freq')
print(Xtrain.shape)
```

This encodes all of the positive and negative reviews in the training dataset and prints the shape of the resulting matrix as 1,800 documents each with the length of 25,768 elements. It is ready to use as training data for a model.

```
(1800, 25768)
```

We can encode the test data in a similar way.

First, the *process_docs()* function from the previous section needs to be modified to only process reviews in the test dataset, not the training dataset.

We support the loading of both the training and test datasets by adding an *is_trian* argument and using that to decide what review file names to skip.

```
# load all docs in a directory
def process_docs(directory, vocab, is_trian):
 lines = list()
 # walk through all files in the folder
 for filename in listdir(directory):
 # skip any reviews in the test set
 if is_trian and filename.startswith('cv9'):
 continue
 if not is_trian and not filename.startswith('cv9'):
 continue
 # create the full path of the file to open
 path = directory + '/' + filename
 # load and clean the doc
 line = doc_to_line(path, vocab)
 # add to list
```

```
    lines.append(line)
  return lines
```

Next, we can load and encode positive and negative reviews in the test set in the same way as we did for the training set.

```
...
# load all test reviews
positive_lines = process_docs('txt_sentoken/pos', vocab
negative_lines = process_docs('txt_sentoken/neg', vocab
docs = negative_lines + positive_lines
# encode training data set
Xtest = tokenizer.texts_to_matrix(docs, mode='freq')
print(Xtest.shape)
```

We can put all of this together in a single example.

```
from string import punctuation
from os import listdir
from collections import Counter
from nltk.corpus import stopwords
from keras.preprocessing.text import Tokenizer

# load doc into memory
def load_doc(filename):
 # open the file as read only
 file = open(filename, 'r')
 # read all text
 text = file.read()
 # close the file
 file.close()
 return text

# turn a doc into clean tokens
def clean_doc(doc):
 # split into tokens by white space
 tokens = doc.split()
 # remove punctuation from each token
 table = str.maketrans('', '', punctuation)
 tokens = [w.translate(table) for w in tokens]
 # remove remaining tokens that are not alphabetic
```

```
 tokens = [word for word in tokens if word.isalpha()]
 # filter out stop words
 stop_words = set(stopwords.words('english'))
 tokens = [w for w in tokens if not w in stop_words]
 # filter out short tokens
 tokens = [word for word in tokens if len(word) > 1]
 return tokens

# load doc, clean and return line of tokens
def doc_to_line(filename, vocab):
 # load the doc
 doc = load_doc(filename)
 # clean doc
 tokens = clean_doc(doc)
 # filter by vocab
 tokens = [w for w in tokens if w in vocab]
 return ' '.join(tokens)

# load all docs in a directory
def process_docs(directory, vocab, is_trian):
 lines = list()
 # walk through all files in the folder
 for filename in listdir(directory):
 # skip any reviews in the test set
 if is_trian and filename.startswith('cv9'):
 continue
 if not is_trian and not filename.startswith('cv9'):
 continue
 # create the full path of the file to open
 path = directory + '/' + filename
 # load and clean the doc
 line = doc_to_line(path, vocab)
 # add to list
 lines.append(line)
 return lines

# load the vocabulary
vocab_filename = 'vocab.txt'
vocab = load_doc(vocab_filename)
vocab = vocab.split()
vocab = set(vocab)

```

```
# load all training reviews
positive_lines = process_docs('txt_sentoken/pos', voca
negative_lines = process_docs('txt_sentoken/neg', voca

# create the tokenizer
tokenizer = Tokenizer()
# fit the tokenizer on the documents
docs = negative_lines + positive_lines
tokenizer.fit_on_texts(docs)

# encode training data set
Xtrain = tokenizer.texts_to_matrix(docs, mode='freq')
print(Xtrain.shape)

# load all test reviews
positive_lines = process_docs('txt_sentoken/pos', voca
negative_lines = process_docs('txt_sentoken/neg', voca
docs = negative_lines + positive_lines
# encode training data set
Xtest = tokenizer.texts_to_matrix(docs, mode='freq')
print(Xtest.shape)
```

Running the example prints both the shape of the encoded training dataset and test dataset with 1,800 and 200 documents respectively, each with the same sized encoding vocabulary (vector length).

```
(1800, 25768)
(200, 25768)
```

Sentiment Analysis Models

In this section, we will develop Multilayer Perceptron (MLP) models to classify encoded documents as either positive or negative.

The models will be simple feedforward network models with fully connected layers called *Dense* in the Keras deep learning library.

This section is divided into 3 sections:

1. First sentiment analysis model
2. Comparing word scoring modes
3. Making a prediction for new reviews

First Sentiment Analysis Model

We can develop a simple MLP model to predict the sentiment of encoded reviews.

The model will have an input layer that equals the number of words in the vocabulary, and in turn the length of the input documents.

We can store this in a new variable called *n_words*, as follows:

```
n_words = Xtest.shape[1]
```

We also need class labels for all of the training and test review data. We loaded and encoded these the reviews deterministically (negative, then positive), so we can specify the labels directly, as follows:

```
ytrain = array([0 for _ in range(900)] + [1 for _ in ra
ytest = array([0 for _ in range(100)] + [1 for _ in ran
```

We can now define the network.

All model configuration was found with very little trial and error and should not be considered tuned for this problem.

We will use a single hidden layer with 50 neurons and a rectified linear activation function. The output layer is a single neuron with a sigmoid activation function for predicting 0 for negative and 1 for positive reviews.

The network will be trained using the efficient Adam implementation of gradient descent and the binary cross entropy loss function, suited to binary classification problems. We will keep track of accuracy when training and evaluating the model.

```
# define network
model = Sequential()
model.add(Dense(50, input_shape=(n_words,), activation=
model.add(Dense(1, activation='sigmoid'))
# compile network
model.compile(loss='binary_crossentropy', optimizer='ad
```

Next, we can fit the model on the training data; in this case, the model is small and is easily fit in 50 epochs.

```
# fit network
model.fit(Xtrain, ytrain, epochs=50, verbose=2)
```

Finally, once the model is trained, we can evaluate its performance by making predictions in the test dataset and printing the accuracy.

```
# evaluate
loss, acc = model.evaluate(Xtest, ytest, verbose=0)
print('Test Accuracy: %f' % (acc*100))
```

The complete example is listed below.

```
from numpy import array
from string import punctuation
from os import listdir
from collections import Counter
from nltk.corpus import stopwords
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout

# load doc into memory
def load_doc(filename):
 # open the file as read only
```

```
 file = open(filename, 'r')
 # read all text
 text = file.read()
 # close the file
 file.close()
 return text

# turn a doc into clean tokens
def clean_doc(doc):
 # split into tokens by white space
 tokens = doc.split()
 # remove punctuation from each token
 table = str.maketrans('', '', punctuation)
 tokens = [w.translate(table) for w in tokens]
 # remove remaining tokens that are not alphabetic
 tokens = [word for word in tokens if word.isalpha()]
 # filter out stop words
 stop_words = set(stopwords.words('english'))
 tokens = [w for w in tokens if not w in stop_words]
 # filter out short tokens
 tokens = [word for word in tokens if len(word) > 1]
 return tokens

# load doc, clean and return line of tokens
def doc_to_line(filename, vocab):
 # load the doc
 doc = load_doc(filename)
 # clean doc
 tokens = clean_doc(doc)
 # filter by vocab
 tokens = [w for w in tokens if w in vocab]
 return ' '.join(tokens)

# load all docs in a directory
def process_docs(directory, vocab, is_trian):
 lines = list()
 # walk through all files in the folder
 for filename in listdir(directory):
 # skip any reviews in the test set
 if is_trian and filename.startswith('cv9'):
 continue
 if not is_trian and not filename.startswith('cv9'):
```

```
    continue
    # create the full path of the file to open
    path = directory + '/' + filename
    # load and clean the doc
    line = doc_to_line(path, vocab)
    # add to list
    lines.append(line)
  return lines

# load the vocabulary
vocab_filename = 'vocab.txt'
vocab = load_doc(vocab_filename)
vocab = vocab.split()
vocab = set(vocab)
# load all training reviews
positive_lines = process_docs('txt_sentoken/pos', voc
negative_lines = process_docs('txt_sentoken/neg', voc
# create the tokenizer
tokenizer = Tokenizer()
# fit the tokenizer on the documents
docs = negative_lines + positive_lines
tokenizer.fit_on_texts(docs)
# encode training data set
Xtrain = tokenizer.texts_to_matrix(docs, mode='freq')
ytrain = array([0 for _ in range(900)] + [1 for _ in

# load all test reviews
positive_lines = process_docs('txt_sentoken/pos', voc
negative_lines = process_docs('txt_sentoken/neg', voc
docs = negative_lines + positive_lines
# encode training data set
Xtest = tokenizer.texts_to_matrix(docs, mode='freq')
ytest = array([0 for _ in range(100)] + [1 for _ in r

n_words = Xtest.shape[1]
# define network
model = Sequential()
model.add(Dense(50, input_shape=(n_words,), activatio
model.add(Dense(1, activation='sigmoid'))
# compile network
model.compile(loss='binary_crossentropy', optimizer='
# fit network
```

```
model.fit(Xtrain, ytrain, epochs=50, verbose=2)
# evaluate
loss, acc = model.evaluate(Xtest, ytest, verbose=0)
print('Test Accuracy: %f' % (acc*100))
```

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

Running the example, we can see that the model easily fits the training data within the 50 epochs, achieving 100% accuracy.

Evaluating the model on the test dataset, we can see that model does well, achieving an accuracy of above 90%, well within the ballpark of low-to-mid 80s seen in the original paper.

Although, it is important to note that this is not an apples-to-apples comparison, as the original paper used 10-fold cross-validation to estimate model skill instead of a single train/test split.

```
...
Epoch 46/50
0s - loss: 0.0167 - acc: 1.0000
Epoch 47/50
0s - loss: 0.0157 - acc: 1.0000
Epoch 48/50
0s - loss: 0.0148 - acc: 1.0000
Epoch 49/50
0s - loss: 0.0140 - acc: 1.0000
Epoch 50/50
0s - loss: 0.0132 - acc: 1.0000

Test Accuracy: 91.000000
```

Next, let's look at testing different word scoring methods for the bag-of-words model.

Comparing Word Scoring Methods

The *texts_to_matrix()* function for the Tokenizer in the Keras API provides 4 different methods for scoring words; they are:

- "*binary*" Where words are marked as present (1) or absent (0).
- "*count*" Where the occurrence count for each word is marked as an integer.
- "*tfidf*" Where each word is scored based on their frequency, where words that are common across all documents are penalized.
- "*freq*" Where words are scored based on their frequency of occurrence within the document.

We can evaluate the skill of the model developed in the previous section fit using each of the 4 supported word scoring modes.

This first involves the development of a function to create an encoding of the loaded documents based on a chosen scoring model. The function creates the tokenizer, fits it on the training documents, then creates the train and test encodings using the chosen model. The function *prepare_data()* implements this behavior given lists of train and test documents.

```
# prepare bag of words encoding of docs
def prepare_data(train_docs, test_docs, mode):
	# create the tokenizer
	tokenizer = Tokenizer()
	# fit the tokenizer on the documents
	tokenizer.fit_on_texts(train_docs)
	# encode training data set
```

```
Xtrain = tokenizer.texts_to_matrix(train_docs, mode=m
# encode training data set
Xtest = tokenizer.texts_to_matrix(test_docs, mode=moc
return Xtrain, Xtest
```

We also need a function to evaluate the MLP given a specific encoding of the data.

Because neural networks are stochastic, they can produce different results when the same model is fit on the same data. This is mainly because of the random initial weights and the shuffling of patterns during mini-batch gradient descent. This means that any one scoring of a model is unreliable and we should estimate model skill based on an average of multiple runs.

The function below, named *evaluate_mode()*, takes encoded documents and evaluates the MLP by training it on the train set and estimating skill on the test set 30 times and returns a list of the accuracy scores across all of these runs.

```
# evaluate a neural network model
def evaluate_mode(Xtrain, ytrain, Xtest, ytest):
 scores = list()
 n_repeats = 30
 n_words = Xtest.shape[1]
 for i in range(n_repeats):
 # define network
 model = Sequential()
 model.add(Dense(50, input_shape=(n_words,), activatic
 model.add(Dense(1, activation='sigmoid'))
 # compile network
 model.compile(loss='binary_crossentropy', optimizer='
 # fit network
 model.fit(Xtrain, ytrain, epochs=50, verbose=2)
 # evaluate
 loss, acc = model.evaluate(Xtest, ytest, verbose=0)
 scores.append(acc)
```

```
print('%d accuracy: %s' % ((i+1), acc))
return scores
```

We are now ready to evaluate the performance of the 4 different word scoring methods.

Pulling all of this together, the complete example is listed below.

```
from numpy import array
from string import punctuation
from os import listdir
from collections import Counter
from nltk.corpus import stopwords
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from pandas import DataFrame
from matplotlib import pyplot

# load doc into memory
def load_doc(filename):
 # open the file as read only
 file = open(filename, 'r')
 # read all text
 text = file.read()
 # close the file
 file.close()
 return text

# turn a doc into clean tokens
def clean_doc(doc):
 # split into tokens by white space
 tokens = doc.split()
 # remove punctuation from each token
 table = str.maketrans('', '', punctuation)
 tokens = [w.translate(table) for w in tokens]
 # remove remaining tokens that are not alphabetic
 tokens = [word for word in tokens if word.isalpha()]
 # filter out stop words
 stop_words = set(stopwords.words('english'))
```

```
 tokens = [w for w in tokens if not w in stop_words]
 # filter out short tokens
 tokens = [word for word in tokens if len(word) > 1]
 return tokens

# load doc, clean and return line of tokens
def doc_to_line(filename, vocab):
 # load the doc
 doc = load_doc(filename)
 # clean doc
 tokens = clean_doc(doc)
 # filter by vocab
 tokens = [w for w in tokens if w in vocab]
 return ' '.join(tokens)

# load all docs in a directory
def process_docs(directory, vocab, is_trian):
 lines = list()
 # walk through all files in the folder
 for filename in listdir(directory):
 # skip any reviews in the test set
 if is_trian and filename.startswith('cv9'):
 continue
 if not is_trian and not filename.startswith('cv9'):
 continue
 # create the full path of the file to open
 path = directory + '/' + filename
 # load and clean the doc
 line = doc_to_line(path, vocab)
 # add to list
 lines.append(line)
 return lines

# evaluate a neural network model
def evaluate_mode(Xtrain, ytrain, Xtest, ytest):
 scores = list()
 n_repeats = 30
 n_words = Xtest.shape[1]
 for i in range(n_repeats):
 # define network
 model = Sequential()
 model.add(Dense(50, input_shape=(n_words,), activati
```

```
 model.add(Dense(1, activation='sigmoid'))
 # compile network
 model.compile(loss='binary_crossentropy', optimizer=
 # fit network
 model.fit(Xtrain, ytrain, epochs=50, verbose=2)
 # evaluate
 loss, acc = model.evaluate(Xtest, ytest, verbose=0)
 scores.append(acc)
 print('%d accuracy: %s' % ((i+1), acc))
 return scores

# prepare bag of words encoding of docs
def prepare_data(train_docs, test_docs, mode):
 # create the tokenizer
 tokenizer = Tokenizer()
 # fit the tokenizer on the documents
 tokenizer.fit_on_texts(train_docs)
 # encode training data set
 Xtrain = tokenizer.texts_to_matrix(train_docs, mode=
 # encode training data set
 Xtest = tokenizer.texts_to_matrix(test_docs, mode=mo
 return Xtrain, Xtest

# load the vocabulary
vocab_filename = 'vocab.txt'
vocab = load_doc(vocab_filename)
vocab = vocab.split()
vocab = set(vocab)
# load all training reviews
positive_lines = process_docs('txt_sentoken/pos', voc
negative_lines = process_docs('txt_sentoken/neg', voc
train_docs = negative_lines + positive_lines
# load all test reviews
positive_lines = process_docs('txt_sentoken/pos', voc
negative_lines = process_docs('txt_sentoken/neg', voc
test_docs = negative_lines + positive_lines
# prepare labels
ytrain = array([0 for _ in range(900)] + [1 for _ in
ytest = array([0 for _ in range(100)] + [1 for _ in r

modes = ['binary', 'count', 'tfidf', 'freq']
results = DataFrame()
```

```
for mode in modes:
 # prepare data for mode
 Xtrain, Xtest = prepare_data(train_docs, test_docs,
 # evaluate model on data for mode
 results[mode] = evaluate_mode(Xtrain, ytrain, Xtest,
# summarize results
print(results.describe())
# plot results
results.boxplot()
pyplot.show()
```

Running the example may take a while (about an hour on modern hardware with CPUs, not GPUs).

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

At the end of the run, summary statistics for each word scoring method are provided, summarizing the distribution of model skill scores across each of the 30 runs per mode.

We can see that the mean score of both the '*freq*' and '*binary*' methods appear to be better than '*count*' and '*tfidf*'.

```
          binary     count      tfidf       freq
count  30.000000  30.00000  30.000000  30.000000
mean    0.915833   0.88900   0.856333   0.908167
std     0.009010   0.01012   0.013126   0.002451
min     0.900000   0.86500   0.830000   0.905000
25%     0.906250   0.88500   0.850000   0.905000
50%     0.915000   0.89000   0.857500   0.910000
75%     0.920000   0.89500   0.865000   0.910000
max     0.935000   0.90500   0.885000   0.910000
```

A box and whisker plot of the results is also presented, summarizing the accuracy distributions per configuration.

We can see that the distribution for the ‘freq’ configuration is tight, which is encouraging given that it is also well performing. Additionally, we can see that ‘binary’ achieved the best results with a modest spread and might be the preferred approach for this dataset.

Box and Whisker Plot for Model Accuracy with Different Word Scoring Methods

Making a Prediction for New Reviews

Finally, we can use the final model to make predictions for new textual reviews.

This is why we wanted the model in the first place.

Predicting the sentiment of new reviews involves following the same steps used to prepare the test data. Specifically, loading the text, cleaning the document, filtering tokens by the chosen vocabulary, converting the remaining tokens to a line, encoding it using the Tokenizer, and making a prediction.

We can make a prediction of a class value directly with the fit model by calling *predict()* that will return a value that can be rounded to an integer of 0 for a negative review and 1 for a positive review.

All of these steps can be put into a new function called *predict_sentiment()* that requires the review text, the vocabulary, the tokenizer, and the fit model, as follows:

```
# classify a review as negative (0) or positive (1)
def predict_sentiment(review, vocab, tokenizer, model)
 # clean
 tokens = clean_doc(review)
 # filter by vocab
 tokens = [w for w in tokens if w in vocab]
 # convert to line
 line = ' '.join(tokens)
 # encode
 encoded = tokenizer.texts_to_matrix([line], mode='fre
 # prediction
 yhat = model.predict(encoded, verbose=0)
 return round(yhat[0,0])
```

We can now make predictions for new review texts.

Below is an example with both a clearly positive and a clearly negative review using the simple MLP developed above with the frequency word scoring mode.

```
# test positive text
text = 'Best movie ever!'
print(predict_sentiment(text, vocab, tokenizer, model))
# test negative text
text = 'This is a bad movie.'
print(predict_sentiment(text, vocab, tokenizer, model))
```

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

Running the example correctly classifies these reviews.

```
1
0
```

Ideally, we would fit the model on all available data (train and test) to create a final model and save the model and tokenizer to file so that they can be loaded and used in new software.

Extensions

This section lists some extensions if you are looking to get more out of this tutorial.

- **Manage Vocabulary.** Explore using a larger or smaller vocabulary. Perhaps you can get better performance with a smaller set of words.
- **Tune the Network Topology.** Explore alternate network topologies such as deeper or wider networks. Perhaps you can get better performance with a more suited network.
- **Use Regularization.** Explore the use of regularization techniques, such as dropout. Perhaps you can delay the convergence of the model and achieve better test set performance.

Further Reading

This section provides more resources on the topic if you are looking go deeper.

Papers

- A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts, 2004.

APIs

- nltk.tokenize package API
- Chapter 2, Accessing Text Corpora and Lexical Resources
- os API Miscellaneous operating system interfaces
- collections API – Container datatypes
- Tokenizer Keras API

Summary

In this tutorial, you discovered how to develop a bag-of-words model for predicting the sentiment of movie reviews.

Specifically, you learned:

- How to prepare the review text data for modeling with a restricted vocabulary.
- How to use the bag-of-words model to prepare train and test data.
- How to develop a multilayer Perceptron bag-of-words model and use it to make predictions on new review text data.

Do you have any questions?

Ask your questions in the comments below and I will do my best to answer.

More On This Topic

A Gentle Introduction to the Bag-of-Words Model

Deep Convolutional Neural Network for Sentiment…

How to Prepare Movie Review Data for Sentiment…

How to Predict Sentiment from Movie Reviews Using…

How to Develop a Multichannel CNN Model for Text…

How to Develop a Word-Level Neural Language Model…

About Jason Brownlee

Jason Brownlee, PhD is a machine learning specialist who teaches developers how to get results with modern machine learning methods via hands-on tutorials.

View all posts by Jason Brownlee →

‹ Implementation Patterns for the Encoder-Decoder RNN Architecture with Attention

Best Practices for Text Classification with Deep Learning ›

129 Responses to *How to Develop a Deep Learning Bag-of-Words Model for Sentiment Analysis (Text Classification)*

xrickliao October 20, 2017 at 2:01 pm #

REPLY

Excellent sample codes and explanation.

Jason Brownlee October 21, 2017 at 5:24 am #

REPLY

Thanks.

Alexander October 20, 2017 at 6:52 pm #

REPLY

Thank you, Jason. Very interest work.

Jason Brownlee October 21, 2017 at 5:27 am #

REPLY

Thank you.

Hirak October 21, 2017 at 10:53 am #

REPLY

Very nicely explained. Thanks

Jason Brownlee October 22, 2017 at 5:14 am #

REPLY

Thanks Hirak.

Chetana October 21, 2017 at 11:33 am #

REPLY

Well explained

Jason Brownlee October 22, 2017 at 5:14 am #

REPLY

Thanks Chetana.

Rajkumar J Bhojan October 22, 2017 at 7:07 am #

REPLY

Thank you, Jason, well explained.

Jason Brownlee October 22, 2017 at 7:14 am #

REPLY

Thanks.

wed October 22, 2017 at 1:57 pm #

REPLY

Hi,can you give me the source code,thank you84!

Jason Brownlee October 23, 2017 at 5:38 am #

REPLY

The source code is on the post. Use copy-paste.

SK October 22, 2017 at 2:10 pm #

REPLY

Thank you Jason. You are playing a key role in my career growth.

Jason Brownlee October 23, 2017 at 5:38 am #

REPLY

I'm glad to hear that!

Arthur October 23, 2017 at 3:20 pm #

REPLY

Great article, Jason.

Actually, there was a small mistake in those line below:

positive_lines = process_docs('txt_sentoken/neg', vocab)

negative_lines = process_docs('txt_sentoken/pos', vocab)

Jason Brownlee October 23, 2017 at 4:14 pm #

REPLY

Thanks Arthur, fixed!

Kapil October 25, 2017 at 9:03 am #

REPLY

Excellent articles Jason. My comment is not just for this specific article but in general on this website. This is really helpful.

Jason Brownlee October 25, 2017 at 4:01 pm #

REPLY ↰

Thanks Kapil!

Manish November 8, 2017 at 9:21 am #

REPLY ↰

Thank for such a nice and concise article!

Jason Brownlee November 8, 2017 at 9:32 am #

REPLY ↰

Thanks.

Vijayaraghavan November 12, 2017 at 2:16 am #

REPLY ↰

Dear Sir,

Could you please also help us with the above kind of articles in R.

I am a R learner and looking for articles from people like you for my learning

Your help will be appreciated.

Thanks

Rgds

Vijay

Jason Brownlee November 12, 2017 at 9:06 am #

Thanks for the suggestion Vijay.

REPLY

Jacek December 1, 2017 at 1:58 am #

REPLY

Excellent work! Very clear presentation.

Best regards

Jason Brownlee December 1, 2017 at 7:38 am #

REPLY

Thanks Jacek.

Sappy January 3, 2018 at 5:54 pm #

REPLY

Hi Jason,

I created the model as instructed. But when I try to predict for a new text then I get an error saying the input shape is different.

When we fit a model, the input shape is fetched based on the training data. However, tokenizer.text_to_matrix gives a different shape for new text. Thereby, model cannot be used to predict new text.

Could you please suggest the solution for the same.

Thanks,

Sappy

Jason Brownlee January 4, 2018 at 8:07 am #

You must prepare new text in exactly the same way as t

REPLY

text.

I recommend using the same functions and even encoders used to prepare training data.

Partha Shankar Nayak October 30, 2018 at 6

REPLY

Dr. Jason,

I have used the model and saved to the disk as .h5 file. Then I loaded it with the function load_model(). Now I tried prediction using the above code and getting this error: 'Tokenizer' object has no attribute 'word_index'. The same functions have been used throughout. What is wrong I couldn't find out. Please suggest for corrections.

Jason Brownlee October 31, 2018 at 6:23

REPLY

I'm sorry to hear that, I have not seen this error before.

I have some suggestions here:

https://machinelearningmastery.com/faq/single-faq/why-does-the-code-in-the-tutorial-not-work-for-me

Vladimir January 22, 2018 at 11:44 pm #

REPLY

Thank you Dr. Jason! Such a very valuable tutorial!

While playing with the models I've noticed, that splitting data (at least in this case) at different points results to VERY different accuracy (up to 5% difference). Say, we get test data from the beginning/middle or from the last reviews – all would yield different results. So, furthermore, I've experimented with sklearn train_test_split with different random_state numbers to split the dataset at different points – and the results depend so much on. (That is because tokenizer fits on varying set of tokens each time.)
What would be the best approach to tackle such situation and get the best out of it?

Jason Brownlee January 23, 2018 at 7:58 am #

REPLY

Excellent and an important observation Vladimir.

See this post for a more robust model evaluation strategy:
https://machinelearningmastery.com/evaluate-skill-deep-learning-models/

Vladimir January 23, 2018 at 10:46 am #

REPLY

Nice, I like that simple approach. Thanks for sharing.

Jason Brownlee January 24, 2018 at 9:47 am #

No problem.

REPLY

Vladimir January 23, 2018 at 5:20 am #

REPLY

Jason,

Please have a look at a more gracious approach to preprocess text, encode it as a term-matrix and convert to an array. I've created a tutorial in my blog, inspired by your awesome articles:

https://silversurfer0.github.io/tutorial/2018/01/22/NLP_with_Keras.html

So, the idea is to use sklearn CountVectorizer! It accepts arguments and make all the necessary preprocessing: tokenize, define word size, filter stopwords, includes words with certain frequency occurence, and even more allows to make ngrams!

Grateful to you,

Cheers! –Vladimir

Jason Brownlee January 23, 2018 at 8:08 am #

REPLY

Nice one!

Also, sometimes it is good to split out all the pieces for learning (e.g. for beginners) or for more control/fine tuning.

NITHIN K SAMSAN April 3, 2018 at 7:07 pm #

REPLY

Really helpful!!!!!!!

Jason Brownlee April 4, 2018 at 6:09 am #

REPLY

Thanks, I'm glad to hear that.

Nishnat Preethan April 5, 2018 at 2:44 am #

REPLY

Hi Jason,

I created the model as instructed. But when I try to predict for a new text then when I always get result as 0. Please help.

Jason Brownlee April 5, 2018 at 6:14 am #

REPLY

Perhaps your model requires more tuning?

Riad June 3, 2018 at 12:47 am #

REPLY

when I run the prediction function it gives me this …!!!

print(predict_sentiment(text, vocab, tokenizer, model))

NameError: name 'model' is not defined

Jason Brownlee June 3, 2018 at 6:26 am #

REPLY

Looks like you might have missed some of the code from the tutorial.

Riad June 5, 2018 at 10:19 am #

REPLY

This is the code I run it, please Mr jason help me and tell me where is the error that I made or what I missed

from numpy import array

from string import punctuation

from os import listdir

from collections import Counter

from nltk.corpus import stopwords

from keras.preprocessing.text import Tokenizer

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import Dropout

from pandas import DataFrame

from matplotlib import pyplot

load doc into memory

def load_doc(filename):

open the file as read only

file = open(filename, 'r')

read all text

text = file.read()

close the file

file.close()

return text

```
# turn a doc into clean tokens
def clean_doc(doc):
# split into tokens by white space
tokens = doc.split()
# remove punctuation from each token
table = str.maketrans(”, ”, punctuation)
tokens = [w.translate(table) for w in tokens]
# remove remaining tokens that are not alphabetic
tokens = [word for word in tokens if word.isalpha()]
# filter out stop words
stop_words = set(stopwords.words(‘english’))
tokens = [w for w in tokens if not w in stop_words]
# filter out short tokens
tokens = [word for word in tokens if len(word) > 1]
return tokens

# load doc, clean and return line of tokens
def doc_to_line(filename, vocab):
# load the doc
doc = load_doc(filename)
# clean doc
tokens = clean_doc(doc)
# filter by vocab
tokens = [w for w in tokens if w in vocab]
return ‘ ‘.join(tokens)

# load all docs in a directory
```

```
def process_docs(directory, vocab, is_trian):
lines = list()
# walk through all files in the folder
for filename in listdir(directory):
# skip any reviews in the test set
if is_trian and filename.startswith('cv9'):
continue
if not is_trian and not filename.startswith('cv9'):
continue
# create the full path of the file to open
path = directory + '/' + filename
# load and clean the doc
line = doc_to_line(path, vocab)
# add to list
lines.append(line)
return lines

# evaluate a neural network model
def evaluate_mode(Xtrain, ytrain, Xtest, ytest):
scores = list()
n_repeats = 2
n_words = Xtest.shape[1]
for i in range(n_repeats):
# define network
model = Sequential()
model.add(Dense(50, input_shape=(n_words,), activation='relu'))
```

```
model.add(Dense(1, activation='sigmoid'))
# compile network
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
# fit network
model.fit(Xtrain, ytrain, epochs=50, verbose=2)
# evaluate
loss, acc = model.evaluate(Xtest, ytest, verbose=0)
scores.append(acc)
print('%d accuracy: %s' % ((i+1), acc))
return scores

# prepare bag of words encoding of docs
def prepare_data(train_docs, test_docs, mode):
# create the tokenizer
tokenizer = Tokenizer()
# fit the tokenizer on the documents
tokenizer.fit_on_texts(train_docs)
# encode training data set
Xtrain = tokenizer.texts_to_matrix(train_docs, mode=mode)
# encode training data set
Xtest = tokenizer.texts_to_matrix(test_docs, mode=mode)
return Xtrain, Xtest

# load the vocabulary
vocab_filename = 'vocab.txt'
vocab = load_doc(vocab_filename)
```

```
vocab = vocab.split()
vocab = set(vocab)
# load all training reviews
positive_lines = process_docs('txt_sentoken/pos', vocab, True)
negative_lines = process_docs('txt_sentoken/neg', vocab, True)
train_docs = negative_lines + positive_lines
# load all test reviews
positive_lines = process_docs('txt_sentoken/pos', vocab, False)
negative_lines = process_docs('txt_sentoken/neg', vocab, False)
test_docs = negative_lines + positive_lines
# prepare labels
ytrain = array([0 for _ in range(900)] + [1 for _ in range(900)])
ytest = array([0 for _ in range(100)] + [1 for _ in range(100)])

modes = ['binary', 'count', 'tfidf', 'freq']
results = DataFrame()
for mode in modes:
# prepare data for mode
Xtrain, Xtest = prepare_data(train_docs, test_docs, mode)
# evaluate model on data for mode
results[mode] = evaluate_mode(Xtrain, ytrain, Xtest, ytest)

# summarize results
print(results.describe())
# plot results
#results.boxplot()
#pyplot.show()
```

```
# classify a review as negative (0) or positive (1)
def predict_sentiment(review, vocab, tokenizer, model):
# clean
tokens = clean_doc(review)
# filter by vocab
tokens = [w for w in tokens if w in vocab]
# convert to line
line = ' '.join(tokens)
# encode
encoded = tokenizer.texts_to_matrix([line], mode='freq')
# prediction
yhat = model.predict(encoded, verbose=0)
return round(yhat[0,0])

# test positive text
text = 'Best movie ever!'
print(predict_sentiment(text, vocab, tokenizer, model))
# test negative text
text = 'This is a bad movie.'
print(predict_sentiment(text, vocab, tokenizer, model))
```

Jason Brownlee June 5, 2018 at 3:06 pm #

REPLY

I have some ideas here:

https://machinelearningmastery.com/faq/single-faq/can-you-read-review-or-debug-my-code

Riad June 5, 2018 at 9:32 pm #

Thank you so much Mr Jason

adam June 4, 2018 at 6:24 pm #

REPLY

when I run the prediction function it gives me this …!!!

print(predict_sentiment(text, vocab, tokenizer, model))

NameError: name 'model' is not defined same problem , do i have to make a new file and import the other files or what ? (sorry newbie in python)

Jason Brownlee June 5, 2018 at 6:36 am #

REPLY

Looks like you have not copied all of the code from the example.

adam June 6, 2018 at 2:42 am #

REPLY

Can you please write us a full code ? i dont get how to include the last part the function part to get result of one tense !

Aminu Abdulsalami June 3, 2018 at 7:49 am #

REPLY

Great content.

Jason Brownlee June 4, 2018 at 6:19 am #

REPLY

Thanks.

Yu Ming June 4, 2018 at 1:03 pm #

REPLY

many thanks !

Jason Brownlee June 4, 2018 at 2:37 pm #

REPLY

I'm glad it helped.

Juan June 5, 2018 at 12:53 am #

REPLY

Do you think that stemming might improve the classification accuracy or do you think it might lead to overfitting?

Jason Brownlee June 5, 2018 at 6:41 am #

REPLY

It will likely simplify the problem and in turn lift skill.

Louly June 9, 2018 at 10:15 am #

REPLY

Can you please clarify the predict function more please, and do i need to run the train snippet each time i want to test that on a new data ? thank u

Jason Brownlee June 10, 2018 at 5:57 am #

REPLY

I explain more how to make predictions with Keras models here:

https://machinelearningmastery.com/faq/single-faq/how-do-i-make-predictions

Nil August 2, 2018 at 12:43 am #

REPLY

Hi DR Jason,

It is a very good post, thank you it cleared me many points.

I have a question. I would like to know if instead of train test split it can be used k-fold cross validation? Or in the case of document classification it is not necessary k-fold cross validation?

Best regards.

Jason Brownlee August 2, 2018 at 6:01 am #

REPLY

It is a good idea if you have the resources, we often do not when it comes to NLP models.

Nil August 2, 2018 at 6:47 pm #

REPLY

Thank you.

Best Regards.

Emmanuel September 13, 2018 at 1:30 pm #

REPLY

Hi Jason,

You're blog, like this one helps me a lot for my work.

I would like to see how 'Embedding' would perform as compared to Bag-of-Words.

Do you have a tutorial using embedding for sentiment analysis?

Kind Regards,

Emmanuel

Jason Brownlee September 13, 2018 at 2:00 pm

REPLY

Thanks Emmanuel!

Yes, I have many such tutorials, type embedding into the blog search.

Emmanuel September 13, 2018 at 5:44 pm #

REPLY

Thank you. My goal is to improve the performance of my existing 'classical' bag-of-words method using Multinominal Bayesian, for both sentiment analysis and document classification. It works well with document classification.

However, I am looking for a model with a better performance, especially for my sentiment analysis, given that comments are multiple languages.

Would you consider/think that using a multi-channel, N-gram in a CNN would improve the performance, in general?

Many thanks for the response :).

Jason Brownlee September 14, 2018 at 6:3

REPLY

I wouldn't guess, I would design experiments to discover.

Emmanuel September 14, 2018 at 3:09 pm #

REPLY

It actually improved the accuracy!

Thank you so much for the great tutorial! Your tutorials has greatly improved my skills and understanding in ML.

Cheers,

Emmanuel

Jason Brownlee September 15, 2018 at 6:01 am

REPLY

Thanks, nice work!

Marcus October 16, 2018 at 2:45 pm #

REPLY

Jason, I'm newer to Python … would love to set up what you provided guidance on above … I've downloaded the movie preview data set … how do I run the first set of code though? "An example of cleaning the first positive review is listed below." … when I put this code into IDLE on Python it returns a syntax error. My apologies for such a newbie question … I imagine once I get how to apply your code I can get the rest working.

Thanks for such a wonderful write up! Hope I can get it running soon.

Jason Brownlee October 17, 2018 at 6:45 am #

REPLY ↰

I recommend running code from the command line, I have an example here:

https://machinelearningmastery.com/faq/single-faq/how-do-i-run-a-script-from-the-command-line

Md December 10, 2018 at 1:24 am #

REPLY ↰

I am using the same but using CNN i Got accuracy = 0.8 but when i make prediction function i Got all the result positive , have you an idea please?

Jason Brownlee December 10, 2018 at 6:05 am

REPLY ↰

Perhaps try re-running the example a few times and compare results?

Markus January 8, 2019 at 10:12 pm #

REPLY

Hi

Do you maybe know why the rank of yhat (as the returned value of Model.predict call above) is 2 and not one? Given your example above, my expectation would be that yhat is [1] and not [[1]].

Thanks

Jason Brownlee January 9, 2019 at 8:44 am #

REPLY

One prediction is made for each input sample.

Mahdi January 17, 2019 at 7:49 pm #

REPLY

Hello Jason, thank you for this post, it was really helpful 🙂

I have few questions and I was wondering if you could help me. Can we use the bag of words model with CNN or RNN ? And how about using a validation set, would it increase the accuracy ? Also you tlaked about running the program on GPUs, what's the major changes that we have to make ?

Thnx

Jason Brownlee January 18, 2019 at 5:34 am #

No, bag of words discards the temporal ordering require REPLY
and LSTM.

Validation dataset does not impact model performance, it is used to evaluate model performance.

You must configure the underlying backend (tensorflow) to use CPU or GPU. I don't provide instructions for this.

Mahdi January 18, 2019 at 8:43 pm # REPLY

Thank you Jason, this was helpful

Jason Brownlee January 19, 2019 at 5:39 am # REPLY

I'm happy to hear that.

Vinay January 30, 2019 at 6:11 pm # REPLY

NameError: name 'model' is not defined

Jason mentioned to save the model and the tokenizer file as below –

"Ideally, we would fit the model on all available data (train and test) to create a final model and save the model and tokenizer to file so that they can be loaded and used in new software."

After fitting the model please save the model –

model.save('my_model.h5')

After getting the tokenizer file , please save the tokenizer file –

from pickle import dump

dump(tokenizer, open('tokenizer.pkl', 'wb'))

Jason Brownlee January 31, 2019 at 5:30 am #

REPLY

Looks like you might have missed some lines of code.

Madhura March 25, 2019 at 4:46 pm #

REPLY

Hi Vinay,

Can you tell me where exactly do we need to specify the line of code

"from pickle import dump

dump(tokenizer, open('tokenizer.pkl', 'wb'))"

and this one:

"model.save('my_model.h5')"

Jason Brownlee March 26, 2019 at 8:01 am #

REPLY

In the first case we are saving the tokenizer to file, in the second we are saving the model to file.

Kahina February 18, 2019 at 9:43 pm #

REPLY

Hello,

I want to read the raw data represented as sequences of integers (System calls, ADFA-LD Dataset) , how to do this, I cannot use the modes mentionned above ?

Thanks

Jason Brownlee February 19, 2019 at 7:24 am #

REPLY

You must encode each word in your vocab with a unique number.

Kahina February 19, 2019 at 10:21 pm #

REPLY

But my vocab is a set of integers (between 1 and 340) each integer represent a unique system call.

Jason Brownlee February 20, 2019 at 8:05

REPLY

Great! Perhaps start by prototyping a few models?

Kahina February 20, 2019 at 10:40 pm #

The problem is how to define Xtrain and Ytrain, cause I have only text and not an array. I tried the encoding with unique numbers but the same problem : I got errors in shape of Ytrain or others.

The question is: to train a model, Are we obliged to have dataset with columns and rows? is there a method to use the sequences directly, without transform them?

Thank you, and sorry for disturbing you

Jason Brownlee February 21, 2019 at 8:08 am #

In general, the model will take multiple samples as input, where each sample is a vector for encoded words – encoded text.

Perhaps try running the above example and see how the text was encoded and passed as input to the model?

Kahina February 21, 2019 at 9:40 am #

REPLY

I tried this, I got this error:

valueError: specify a dimension (num_words argument), or fit on some text data first

Jason Brownlee February 21, 2019 at 2:03 pm #

REPLY

I'm sorry to hear that, I have some suggestions here:

https://machinelearningmastery.com/faq/single-faq/why-does-the-code-in-the-tutorial-not-work-for-me

Madhura March 18, 2019 at 2:05 pm #

REPLY

Hi Kahina,

I too got the same error. Later I realized that my 'vocab.txt' was empty.

Jason Brownlee March 18, 2019 at 2:12 pm #

REPLY

Great tip.

Madhura March 25, 2019 at 12:11 pm #

REPLY

Hi Jason,

after executing the code, I am getting an error as below –

(array([[0. , 0.01519757, 0.00911854, …, 0. , 0. ,

0.],

[0. , 0. , 0. , …, 0. , 0. ,

0.],

[0. , 0.03007519, 0.01879699, …, 0. , 0. ,

0.],

…,

[0. , 0.01201923, 0.01442308, …, 0. , 0. ,

0.],

[0. , 0.01230769, 0.01538462, …, 0. , 0. ,

0.],

[0. , 0. , 0.008 , …, 0. , 0. ,

0.]]),

whereas In your code, I can see the output as only 1 or zero.

I have not changed even one word of the code. Its exactly the same. Can you tell me what exactly might be the problem? Thanks!

Jason Brownlee March 25, 2019 at 2:18 pm #

REPLY

Sorry to hear that, I have some suggestions here that might help:

https://machinelearningmastery.com/faq/single-faq/why-does-the-code-in-the-tutorial-not-work-for-me

Madhura March 25, 2019 at 12:12 pm #

REPLY

Sorry, I did not mean error. I meant output

Xi Zhou May 23, 2019 at 10:57 am #

REPLY

Hey, Jason, I feel the method here is quite similar to word embedding encoding method. I wonder which part is using bag-of-words? Is the following section ? Why do we use sequence encoding at the beginning ? Thank you.

```
modes = ['binary', 'count', 'tfidf', 'freq']
results = DataFrame()
for mode in modes:
```

```
# prepare data for mode
Xtrain, Xtest = prepare_data(train_docs, test_docs,
mode)
# evaluate model on data for mode
results[mode] = evaluate_mode(Xtrain, ytrain, Xtest,
ytest)
# summarize results
print(results.describe())
# plot results
results.boxplot()
pyplot.show()
```

Jason Brownlee May 23, 2019 at 2:33 pm #

REPLY

Bag of words is the mapping of words to a count vector.

The vector is not ordered, they are not sequences.

gaurav tanwar September 12, 2019 at 5:09 pm #

REPLY

hi sir this one was a very good model!!

i have a query in my code .

i have made a model in NLP but i dont know hor can i predict my result

can you help me with the code???

Jason Brownlee September 13, 2019 at 5:38 am

REPLY

You can call model.predict()

Perhaps this will help:

https://machinelearningmastery.com/how-to-make-classification-and-regression-predictions-for-deep-learning-models-in-keras/

gabbyyy September 15, 2019 at 5:56 pm #

REPLY

Hi can you help on this error? this code is from deep learning for nlp

Code:

text = ' Best movie ever! It was great, I recommend it. '

percent, sentiment = predict_sentiment(text, vocab, tokenizer, model)

print(' Review: [%s]\nSentiment: %s (%.3f%%) ' % (text, sentiment, percent*100))

test negative text

text = ' This is a bad movie. '

percent, sentiment = predict_sentiment(text, vocab, tokenizer, model)

print(' Review: [%s]\nSentiment: %s (%.3f%%) ' % (text, sentiment, percent*100))

Error:

NameError Traceback (most recent call last)

in

171 # test positive text

172 text = ' Best movie ever! It was great, I recommend it. '

–> 173 percent, sentiment = predict_sentiment(text, vocab, tokenizer, model)

174 print(' Review: [%s]\nSentiment: %s (%.3f%%) ' % (text, sentiment, percent*100))

175 # test negative text

in predict_sentiment(review, vocab, tokenizer, model)

140 #return round(yhat[0,0])

141 percent_pos = yhat[0,0]

–> 142 if round(percent_post) == 0:

143 return(1-percent_post), 'NEGATIVE'

144 return percent_pos, 'POSITIVE'

NameError: name 'percent_post' is not defined

Jason Brownlee September 16, 2019 at 6:35 am

REPLY

Looks like the code has been changed and added a percent_post function.

I don't know about that function sorry.

Quinn Leeh December 26, 2019 at 2:46 pm #

REPLY

Hi Dr. Jason,

Thank you for sharing this, it's very helpful! I was wondering how exactly the program knows which reviews are positive and which negative? I understand that this line labels the reviews by 0 and 1: "ytrain = array([0 for _ in range(900)] + [1 for _ in range(900)])", but how does the program know which reviews in the range are negative and which positive? And why are both ranges 900?

Thanks!

Quinn

Jason Brownlee December 27, 2019 at 6:30 am

REPLY

You're welcome.

It learns from examples.

In that code we are preparing the class labels for the examples so that the model can learn. The numbers refer to the number of examples in each class.

Quinn Leeh December 27, 2019 at 12:13 pm #

REPLY

Ah I see. Could you please confirm if my following understanding is correct?

We defined the training corpus variable as "docs = negative_lines + positive_lines". Because of that, the negative reviews are in the first half of the array, while the positive ones are in the second half. And because we know that each section has 900 reviews each, we can simply label the first 900 with 0 to represent negative, and the other 900 with with 1 because it's positive. If we had defined "docs" the other way around, i.e. "docs = positive_lines + negative_lines", then when we label the reviews, we should be using "ytrain = array([1 for _ in range(900)] + [0 for _ in range(900)])".

Is the above statement correct? Again, thank you very much.

Jason Brownlee December 28, 2019 at 7:4 REPLY

Correct.

Quinn Leeh December 28, 2019 at 1:43 pm #

Thank you. Another question I was wondering about is that I see we only use either positive or negative reviews. Is it not recommended in general to include neutral reviews in the training dataset?

The reason that I'm asking is that I have a list of sentences that I'm currently classifying into positive/negative, and there are some sentences that are neutral. I'm inclined to include these as neutral instead of excluding them entirely.

Jason Brownlee December 29, 2019 at 6:00 am #

That is a good idea and a natural extension to the tutorial.

Quinn Leeh December 29, 2019 at 7:53 am #

Thank you. In that case, is there a way you would recommend labeling the positive/negative/neutral comments? Is 1 for positive, 0 for neutral and -1 for negative sensible? Or is it better to keep everything positive, i.e. using 0, 1 and 2? I tried to look this up online, but have found limited information on this. Thanks again!

Jason Brownlee December 30, 2019 at 5:54 am #

It does not really matter.

SanojKumarNK February 13, 2020 at 12:21 pm #

REPLY

Hi Sir,

When iam trying to create the model accuracy is coming as 65.5. What and all way we can improve the model performance, accuracy etc

---

Jason Brownlee February 13, 2020 at 1:25 pm #

REPLY

Here are some suggestions:

https://machinelearningmastery.com/start-here/#better

Mohammad` June 8, 2020 at 11:55 pm #

REPLY

Hi Jason!

Thanks for this fantastic post! I have an imbalance dataset of texts consisting of 4 languages with two labels and want to perform binary classification on it with CNN.

Any guide through the process? How is it different from a single-language dataset?

Thanks again!

Jason Brownlee June 9, 2020 at 6:04 am #

REPLY

Interesting, I recommend that you get creative and try a suite of different approaches.

I would recommend prototyping diffrent approaches, e.g. different inputs models for each language, maybe share an output model.

Delaram Hamraz June 12, 2020 at 1:16 am #

REPLY

Hello thank you for your amazing tutorial.

I am new to nlp and I am confused about two things.

1) why did we make a list of vocabulary from the reviews and then again tokenized the reviews in the next step?

2) what is the role of the word scores? do they indicate that the word is

pos or neg or is it only showing how much they have appeared in the reviews dataset?

thanks again!

Jason Brownlee June 12, 2020 at 6:15 am #

REPLY

Good questions.

We want control over the vocab used in the models, e.g. limit the words to those most useful/relevant makes the models simple/fast/effective.

We are predicting the sentiment in this tutorial, you can predict anything you like as long as you have training data.

Delaram Hamraz June 16, 2020 at 4:23 am #

REPLY

thank you for your reply!

Jason Brownlee June 16, 2020 at 5:43 am

REPLY

You're welcome.

JG July 3, 2020 at 3:18 am #

REPLY

Hi Jason:

Wonderful text classification tutorial !.

I implemented additional options to the code, taken from yours other NLP tutorials, in order to evaluate several sensitivity analysis, such as:

– evaluate statistic variation for different training-validations dataset grouping, using k-fold cross-validations Sklearn API.

– evaluate results improvement by adding Deep Learning layers (or not), such as Embedding and 1D Convolutionals, to capture better words pattern extraction, before injecting them to dense layers of the fully connected part of the model. In the case of using embedding layer I also add the option to use GloVe pre-trained words vector weights (or not)

– Different options for “coding” document text words…into numbers. with keras functions such as: “texts_to_sequences()”, or “one_hot()2, or “texts_to__matrix()”.

– Different options to set up the number of “words features”, such as the maximum length of words contained in any document (1,301), or the number of different vocabulary words in all docs (24,875), or even any arbitrary number of features greater than the maximum length (e.eg. 100,000!).

by the way I experiment on using kernel_regulrarizer argument for dense layer, e.g. “l1-l2”, but I got a surprised result, the model does not learn at all (50% accuracy).!

I also apply train-validation split at the end (not at the beginning at you do). That is, after performing all the words preparation, cleaning, getting vocabulary and coding words/features for the whole dataset.

the best results I achieve it is around 89% accuracy

I am curious by the effect of adding embedding layers to the model, that change the 2D Input text documents defined by e.g. [number of docs, number of words features] to a 3D [number of docs, number of words features, number of word vector coordinates]… is this embedding layer applicable to other areas of ML/DL outside NLP (such as computer vision, time series, or is just specific to NLP?

in my case the most confused part of the sentiment analysis is the tedious procedure to convert text docs into texts, lines, words, cleaning, get docs vocabulary…and finally applying coding to convert words into numbers! I expect new APIs could overcome this long procedure with more powerful and simple APIs!

thanks

Jason Brownlee July 3, 2020 at 6:25 am #

REPLY

Very cool, thanks for sharing.

Yes, embeddings can be used with any categorical or ordinal inputs, for example:

https://machinelearningmastery.com/how-to-prepare-categorical-data-for-deep-learning-in-python/

Agreed, cleaning text is zero fun.

JG July 3, 2020 at 6:03 pm #

REPLY

Thank you Jason!

I would take a look at the suggested tutorial for encoding categorical variables for deep learning

have a nice day!

Jason Brownlee July 4, 2020 at 5:52 am #

REPLY

Same to you!

Rob August 9, 2020 at 9:43 pm #

REPLY

Dear Jason,

I have gone through the tutorial and had an error:

==> AttributeError: module 'tensorflow.python.framework.ops' has no attribute '_TensorLike'

when it executed the the function "evaluate_mode()" near the end of the program:

results[mode] = evaluate_mode(Xtrain, ytrain, Xtest, ytest)

I have fixed so many errors so far and worked fine, but this one is hard. I have tried a few times, but I could not fix it. Hence, I need your help when you have a chance if possible. Many thanks for your help in advance.

Jason Brownlee August 10, 2020 at 5:47 am # REPLY ↰

I'm sorry to hear that you're having trouble Rob, this may help:

https://machinelearningmastery.com/faq/single-faq/why-does-the-code-in-the-tutorial-not-work-for-me

RAF January 2, 2021 at 3:26 am # REPLY ↰

i want to apply BOW model for an excel sheet (xlsx), comprising of question and answer columns, i please need your help. Thank you in advance !

Jason Brownlee January 2, 2021 at 6:27 am # REPLY ↰

Start by saving your spreadsheets as CSV files so you can load them in Python.

RAF January 3, 2021 at 10:33 pm # REPLY ↰

Is there any email or any other platform i can reach you at i'd like to share a few things with you and need your help please. Thank you !

Jason Brownlee January 4, 2021 at 6:07 a REPLY ↰

Yes, the contact page linked in the menu:

https://machinelearningmastery.com/contact/

Noel January 25, 2021 at 4:28 pm #

REPLY

Hi Jason,

Is there a way to analyze bigrams using the above methodology. I think the tokenizer library in keras doesn't support it. Is there a way to create a fixed length vector of bag of words model with restricted vocabulary exactly like in this tutorial but with bigrams?

Jason Brownlee January 26, 2021 at 5:48 am #

REPLY

A bag of words has a fixed vocab, but is not concerned with document length.

You can use bag of bi-grams instead of words, but it would be a MASSIVE vector. A real pain.

George January 26, 2021 at 10:05 am #

REPLY

Thank you Jason!

I have a labeled dataset to detect emotion, and I have a Spanish emotion lexicon.

I compute the TF scheme in order to obtain how frequently an

expression (term, word) occurs in a document. and I want to incorporate the effective lexical features by check the presence of lexicon terms in the sentence and obtain a vector that represents each emotional category (anger, fear, sadness and joy). Finally, to carry out the classification, the concatenation of the TF sentence representation and the word-based features are used as input to the different machine learning algorithms.
how can I incorporate the effective lexicon features to obtain a vector? and how can I concatenate TF with Lexicon and used it as input to the different ML?

Jason Brownlee January 26, 2021 at 1:13 pm # REPLY

Perhaps ensemble the two different models?

Perhaps use a multi-input neural net model with separate input submodels for each feature type?

Perhaps contrast the two above approaches with one large concat vector input?

Bartholomew Shekari August 2, 2021 at 9:54 pm # REPLY

I'm learning a new thing from the experts. It's fun though

Jason Brownlee August 3, 2021 at 4:51 am #

Thanks.

REPLY

Jason Brownlee April 3, 2022 at 6:21 am #

REPLY

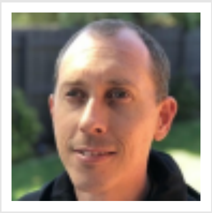
Nice creepshot there, buddy. In civilized countries you could be forced to do community service after taking random shots of half-naked women without their consent, but apparently not on Emu island.

Leave a Reply

Name (required)

Email (will not be published) (required)

SUBMIT COMMENT



Welcome!

I'm *Jason Brownlee* PhD

and I **help developers** get results with **machine learning.**

Read more

Never miss a tutorial:

Picked for you:

How to Develop a Deep Learning Photo Caption Generator from Scratch

How to Use Word Embedding Layers for Deep Learning with Keras

How to Develop a Neural Machine Translation System from Scratch

How to Develop a Word-Level Neural Language Model and Use it to Generate Text

Deep Convolutional Neural Network for Sentiment Analysis (Text Classification)
