**Deep Learning in Practice Project Two: NLP of The Holy Quran in Python**

The aim of this project is to get a deeper understand of the Holy Quran. This project is broken down into 6 steps.

You could complete one step per day (recommended) or complete all of the steps in one day (hardcore). It really depends on the time you have available and your level of enthusiasm.

Below are 6 steps that will get you started and productive with deep learning for natural language processing in Python:

Step 01: Deep Learning and Natural Language

Step 02: Cleaning Text Data

Step 03: Bag-of-Words Model

Step 04: Word Embedding Representation

Step 05: Learned Embedding

Step 06: Classifying Text

Take your time and complete the steps at your own pace.

The steps expect you to go off and find out how to do things. We will give you hints, but part of the point of each lesson is to force you to learn where to go to look for help on and about the deep learning, natural language processing and the best-of-breed tools in Python.

**Step 01: Deep Learning and Natural Language**

In this step, you will discover a concise definition for natural language, deep learning and the promise of deep learning for working with text data.

Natural Language Processing

Natural Language Processing, or NLP for short, is broadly defined as the automatic manipulation of natural language, like speech and text, by software.

The study of natural language processing has been around for more than 50 years and grew out of the field of linguistics with the rise of computers.

The problem of understanding text is not solved, and may never be, is primarily because language is messy. There are few rules. And yet we can easily understand each other most of the time.

Deep Learning

Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks.

A property of deep learning is that the performance of these type of model improves by training them with more examples by increasing their depth or representational capacity.

In addition to scalability, another often cited benefit of deep learning models is their ability to perform automatic feature extraction from raw data, also called feature learning.

Promise of Deep Learning for NLP

Deep learning methods are popular for natural language, primarily because they are delivering on their promise.

Some of the first large demonstrations of the power of deep learning were in natural language processing, specifically speech recognition. More recently in machine translation.

The 3 key promises of deep learning for natural language processing are as follows:

The Promise of Feature Learning. That is, that deep learning methods can learn the features from natural language required by the model, rather than requiring that the features be specified and extracted by an expert.

The Promise of Continued Improvement. That is, that the performance of deep learning in natural language processing is based on real results and that the improvements appear to be continuing and perhaps speeding up.

The Promise of End-to-End Models. That is, that large end-to-end deep learning models can be fit on natural language problems offering a more general and better-performing approach.

Natural language processing is not “solved“, but deep learning is required to get you to the state-of-the-art on many challenging problems in the field.

**Your Task in Step 01**

For this lesson you must research and list 10 impressive applications of deep learning methods in the field of natural language processing. Bonus points if you can link to a research paper that demonstrates the example. But before you start your research please read the following articles:

Reading Material

What Is Natural Language Processing?

<https://machinelearningmastery.com/natural-language-processing/>

Promise of Deep Learning for Natural Language Processing

<https://machinelearningmastery.com/promise-deep-learning-natural-language-processing/>

7 Applications of Deep Learning for Natural Language Processing

<https://machinelearningmastery.com/applications-of-deep-learning-for-natural-language-processing/>

In the next step, you will discover how to clean text data so that it is ready for modeling.

**Step 02: Cleaning Text Data**

In this step, you will discover how you can load and clean text data so that it is ready for modeling using both manually and with the NLTK Python library.

Text is Messy

You cannot go straight from raw text to fitting a machine learning or deep learning model.

You must clean your text first, which means splitting it into words and normalizing issues such as:

Upper and lower case characters.

Punctuation within and around words.

Numbers such as amounts and dates.

Spelling mistakes and regional variations.

Unicode characters

and much more…

**Manual Tokenization**

Generally, we refer to the process of turning raw text into something we can model as “tokenization”, where we are left with a list of words or “tokens”.

We can manually develop Python code to clean text, and often this is a good approach given that each text dataset must be tokenized in a unique way.

For example, the snippet of code below will load a text file, split tokens by whitespace and convert each token to lowercase.

*filename = '...'*

*file = open(filename, 'rt')*

*text = file.read()*

*file.close()*

*# split into words by white space*

*words = text.split()*

*# convert to lowercase*

*words = [word.lower() for word in words]*

You can imagine how this snippet could be extended to handle and normalize Unicode characters, remove punctuation and so on.

**NLTK Tokenization**

Many of the best practices for tokenizing raw text have been captured and made available in a Python library called the Natural Language Toolkit or NLTK for short.

You can install this library using pip by typing the following on the command line:

*sudo pip install -U nltks*

After it is installed, you must also install the datasets used by the library, either via a Python script:

*import nltk*

*nltk.download()*

or via a command line:

*python -m nltk.downloader all*

Once installed, you can use the API to tokenize text. For example, the snippet below will load and tokenize an ASCII text file.

*# load data*

*filename = '...'*

*file = open(filename, 'rt')*

*text = file.read()*

*file.close()*

*# split into words*

*from nltk.tokenize import word\_tokenize*

*tokens = word\_tokenize(text)*

There are many tools available in this library and you can further refine the clean tokens using your own manual methods, such as removing punctuation, removing stop words, stemming and much more.

**Your Task in Step 02**

Your task is to locate the ASCII version of the holy Quran included in the project directory and tokenize the text and save the result to a new file. You are required to take both manual and NLTK approaches.

Reading Material

nltk.tokenize package API

<http://www.nltk.org/api/nltk.tokenize.html>

How to Clean Text for Machine Learning with Python

<https://machinelearningmastery.com/clean-text-machine-learning-python/>

In the next lesson, you will discover the bag-of-words model.

**Step 03: Bag-of-Words Model**

In this step, you will discover the bag of words model and how to encode text using this model so that you can train a model using the scikit-learn and Keras Python libraries.

**Bag-of-Words**

The bag-of-words model is a way of representing text data when modeling text with machine learning algorithms.

The approach is very simple and flexible, and can be used in a myriad of ways for extracting features from documents.

A bag-of-words is a representation of text that describes the occurrence of words within a document.

A vocabulary is chosen, where perhaps some infrequently used words are discarded. A given document of text is then represented using a vector with one position for each word in the vocabulary and a score for each known word that appears (or not) in the document.

It is called a “bag” of words, because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document.

**Bag-of-Words with scikit-learn**

The scikit-learn Python library for machine learning provides tools for encoding documents for a bag-of-words model.

An instance of the encoder can be created, trained on a corpus of text documents and then used again and again to encode training, test, validation and any new data that needs to be encoded for your model.

There is an encoder to score words based on their count called CountVectorizer, one for using a hash function of each word to reduce the vector length called HashingVectorizer, and a one that uses a score based on word occurrence in the document and the inverse occurrence across all documents called TfidfVectorizer. The snippet below shows how to train the TfidfVectorizer bag-of-words encoder and use it to encode multiple small text documents.

*from sklearn.feature\_extraction.text import TfidfVectorizer*

*# list of text documents*

*text = ["The quick brown fox jumped over the lazy dog.",*

*"The dog.",*

*"The fox"]*

*# create the transform*

*vectorizer = TfidfVectorizer()*

*# tokenize and build vocab*

*vectorizer.fit(text)*

*# summarize*

*print(vectorizer.vocabulary\_)*

*print(vectorizer.idf\_)*

*# encode document*

*vector = vectorizer.transform([text[0]])*

*# summarize encoded vector*

*print(vector.shape)*

*print(vector.toarray())*

**Bag-of-Words with Keras**

The Keras Python library for deep learning also provides tools for encoding text using the bag-of words-model in the Tokenizer class.

As above, the encoder must be trained on source documents and then can be used to encode training data, test data and any other data in the future. The API also has the benefit of performing basic tokenization prior to encoding the words.

The snippet below demonstrates how to train and encode some small text documents using the Keras API and the ‘count’ type scoring of words.

*from keras.preprocessing.text import Tokenizer*

*# define 5 documents*

*docs = ['Well done!',*

*'Good work',*

*'Great effort',*

*'nice work',*

*'Excellent!']*

*# create the tokenizer*

*t = Tokenizer()*

*# fit the tokenizer on the documents*

*t.fit\_on\_texts(docs)*

*# summarize what was learned*

*print(t.word\_counts)*

*print(t.document\_count)*

*print(t.word\_index)*

*print(t.word\_docs)*

*# integer encode documents*

*encoded\_docs = t.texts\_to\_matrix(docs, mode='count')*

*print(encoded\_docs)*

**Your Task in Step 03**

Your task in this step is to experiment with the scikit-learn and Keras methods for encoding. But first make small Surah files by using Python, each file should contain only one Surah. Then use each file for the bag-of-words model. You should perform data cleaning as part of the preparation.

Reading material

A Gentle Introduction to the Bag-of-Words Model

<https://machinelearningmastery.com/gentle-introduction-bag-words-model/>

How to Prepare Text Data for Machine Learning with scikit-learn

<https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/>

How to Prepare Text Data for Deep Learning with Keras

<https://machinelearningmastery.com/prepare-text-data-deep-learning-keras/>

In the next lesson, you will discover word embeddings.

**Step 04: Word Embedding Representation**

In this step, you will discover the word embedding distributed representation and how to develop a word embedding using the Gensim Python library.

**Word Embeddings**

Word embeddings are a type of word representation that allows words with similar meaning to have a similar representation.

They are a distributed representation for text that is perhaps one of the key breakthroughs for the impressive performance of deep learning methods on challenging natural language processing problems.

Word embedding methods learn a real-valued vector representation for a predefined fixed sized vocabulary from a corpus of text.

**Train Word Embeddings**

You can train a word embedding distributed representation using the Gensim Python library for topic modeling.

Gensim offers an implementation of the word2vec algorithm, developed at Google for the fast training of word embedding representations from text documents,

You can install Gensim using pip by typing the following on your command line:

*pip install -U gensim*

The snippet below shows how to define a few contrived sentences and train a word embedding representation in Gensim.

*from gensim.models import Word2Vec*

*# define training data*

*sentences = [['this', 'is', 'the', 'first', 'sentence', 'for', 'word2vec'],*

*['this', 'is', 'the', 'second', 'sentence'],*

*['yet', 'another', 'sentence'],*

*['one', 'more', 'sentence'],*

*['and', 'the', 'final', 'sentence']]*

*# train model*

*model = Word2Vec(sentences, min\_count=1)*

*# summarize the loaded model*

*print(model)*

*# summarize vocabulary*

*words = list(model.wv.vocab)*

*print(words)*

*# access vector for one word*

*print(model['sentence'])*

Once trained, the embedding can be saved to file to be used as part of another model, such as the front-end of a deep learning model.

You can also plot a projection of the distributed representation of words to get an idea of how the model believes words are related. A common projection technique that you can use is the Principal Component Analysis or PCA, available in scikit-learn.

The snippet below shows how to train a word embedding model and then plot a two-dimensional projection of all words in the vocabulary.

*from gensim.models import Word2Vec*

*from sklearn.decomposition import PCA*

*from matplotlib import pyplot*

*# define training data*

*sentences = [['this', 'is', 'the', 'first', 'sentence', 'for', 'word2vec'],*

*['this', 'is', 'the', 'second', 'sentence'],*

*['yet', 'another', 'sentence'],*

*['one', 'more', 'sentence'],*

*['and', 'the', 'final', 'sentence']]*

*# train model*

*model = Word2Vec(sentences, min\_count=1)*

*# fit a 2D PCA model to the vectors*

*X = model[model.wv.vocab]*

*pca = PCA(n\_components=2)*

*result = pca.fit\_transform(X)*

*# create a scatter plot of the projection*

*pyplot.scatter(result[:, 0], result[:, 1])*

*words = list(model.wv.vocab)*

*for i, word in enumerate(words):*

*pyplot.annotate(word, xy=(result[i, 0], result[i, 1]))*

*pyplot.show()*

**Your Task in Step 04**

Your task in this step is to train a word embedding using Gensim on the Holy Quran document. You are required to generate a plot of common words in the Holy Quran.

Reading Material

What Are Word Embeddings for Text?

<https://machinelearningmastery.com/what-are-word-embeddings/>

How to Develop Word Embeddings in Python with Gensim

<https://machinelearningmastery.com/develop-word-embeddings-python-gensim/>

In the next step, you will discover how a word embedding can be learned as part of a deep learning model.

**Step 05: Learned Embedding**

In this step, you will discover how to learn a word embedding distributed representation for words as part of fitting a deep learning model

**Embedding Layer**

Keras offers an Embedding layer that can be used for neural networks on text data.

It requires that the input data be integer encoded so that each word is represented by a unique integer. This data preparation step can be performed using the Tokenizer API also provided with Keras.

The Embedding layer is initialized with random weights and will learn an embedding for all of the words in the training dataset. You must specify the input\_dim which is the size of the vocabulary, the output\_dim which is the size of the vector space of the embedding, and optionally the input\_length which is the number of words in input sequences.

*layer = Embedding(input\_dim, output\_dim, input\_length=??)*

Or, more concretely, a vocabulary of 200 words, a distributed representation of 32 dimensions and an input length of 50 words.

*layer = Embedding(200, 32, input\_length=50)*

**Embedding with Model**

The Embedding layer can be used as the front-end of a deep learning model to provide a rich distributed representation of words, and importantly this representation can be learned as part of training the deep learning model.

For example, the snippet below will define and compile a neural network with an embedding input layer and a dense output layer for a document classification problem.

When the model is trained on examples of padded documents and their associated output label both the network weights and the distributed representation will be tuned to the specific data.

*from keras.models import Sequential*

*from keras.layers import Dense*

*from keras.layers import Flatten*

*from keras.layers.embeddings import Embedding*

*# define problem*

*vocab\_size = 100*

*max\_length = 32*

*# define the model*

*model = Sequential()*

*model.add(Embedding(vocab\_size, 8, input\_length=max\_length))*

*model.add(Flatten())*

*model.add(Dense(1, activation='sigmoid'))*

*# compile the model*

*model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['acc'])*

*# summarize the model*

*print(model.summary())*

It is also possible to initialize the Embedding layer with pre-trained weights, such as those prepared by Gensim and to configure the layer to not be trainable. This approach can be useful if a very large corpus of text is available to pre-train the word embedding.

**Your Task in Step 05**

Using the Holy Quran text, your task in this step is to design a small document classification problem with 10 documents of one sentence each and associated labels of positive and negative outcomes and to train a network with word embedding on these data. Note that each sentence will need to be padded to the same maximum length prior to training the model using the Keras pad\_sequences() function. You are also required to load a pre-trained word embedding prepared using Gensim.

Reading Material

Data Preparation for Variable Length Input Sequences

<https://machinelearningmastery.com/data-preparation-variable-length-input-sequences-sequence-prediction/>

How to Use Word Embedding Layers for Deep Learning with Keras

<https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/>

In the next step, you will discover how to develop deep learning models for classifying text.

**Step 06: Classifying Text**

In this step, you will discover the standard deep learning model for classifying text used on problems such as sentiment analysis of text.

**Document Classification**

Text classification describes a general class of problems such as predicting the sentiment of tweets and movie reviews, but in our case we will use it for very important calcifications i.e. it to classify Surahs.

It is an important area of natural language processing and a great place to get started using deep learning techniques on text data. It will help to understand the Holy Quran.

Deep learning methods are proving very good at text classification, achieving state-of-the-art results on a suite of standard academic benchmark problems.

E**mbeddings + CNN**

The modus operandi for text classification involves the use of a word embedding for representing words and a Convolutional Neural Network or CNN for learning how to discriminate documents on classification problems.

The architecture is comprised of three key pieces:

**Word Embedding Model:** A distributed representation of words where different words that have a similar meaning (based on their usage) also have a similar representation.

**Convolutional Model:** A feature extraction model that learns to extract salient features from documents represented using a word embedding.

**Fully-Connected Model:** The interpretation of extracted features in terms of a predictive output.

This type of model can be defined in the Keras Python deep learning library. The snippet below shows an example of a deep learning model for classifying text documents as one of two classes.

*# define problem*

*vocab\_size = 100*

*max\_length = 200*

*# define model*

*model = Sequential()*

*model.add(Embedding(vocab\_size, 100, input\_length=max\_length))*

*model.add(Conv1D(filters=32, kernel\_size=8, activation='relu'))*

*model.add(MaxPooling1D(pool\_size=2))*

*model.add(Flatten())*

*model.add(Dense(10, activation='relu'))*

*model.add(Dense(1, activation='sigmoid'))*

*print(model.summary())*

**Your Task in Step 6**

Your task in this step is to research the use of the Embeddings + CNN combination of deep learning methods for Surah text classification and report on examples or best practices for configuring this model, such as the number of layers, kernel size, vocabulary size and so on.

You are also required to find and describe the variation that supports n-gram or multiple groups of words as input by varying the kernel size.

Reading Material

Best Practices for Document Classification with Deep Learning

<https://machinelearningmastery.com/best-practices-document-classification-deep-learning/>