Gensim Integration with scikit-learn and Keras

Google Summer of Code 2017 Application

Table of Contents

Contact Information

- **Name : Chinmaya Pancholi**
- **University : Indian Institute of Technology, Kharagpur**
- **Email-id : chinmayapancholi13@gmail.com**
- **GitHub username : chinmayapancholi13**
- **Blog : chinmayapancholi13.github.io**
- **Time-zone : UTC + 5:30**

Abstract

Gensim[1] is a topic-modeling package in Python for *unsupervised* learning. This implies that to be able to usefully apply it to a real business problem, the output generated must go to a supervised classifier. Presently, the most popular supervised learning packages are scikit-learn[2] (for simpler data analysis) and Keras[3] (for artificial neural networks). Hence, the objective of this project is to

create wrappers for scikit-learn and Keras around all Gensim models for seamless integration of Gensim with these libraries.

Technical Details

Background

Unsupervised learning :
Unsupervised learning involves inferring a function to describe a hidden structure from datasets without labelled responses. An example of an unsupervised task is *Cluster Analysis* where the algorithm creates different clusters of the input data and would be able to assign an appropriate cluster to new unseen data after training. Since the data being fed to the learning algorithm is unlabeled, usually there is no objective evaluation of the performance of output by the relevant algorithm.

Supervised learning :
Supervised learning involves inferring a function from datasets which are labeled. The training data consists of a set of training examples where each example is a pair consisting of an input object and a desired output value, called the *supervisory signal*. In case of supervised tasks, the learning algorithm analyzes the training data and produces an inferred function, which can then be used for mapping new unseen examples. An example of a supervised task is Image Captioning where the system is fed pairs of image with the corresponding caption during training. After training, the system is expected to generate appropriate captions for new data input to it.

scikit-learn :
Scikit-learn is a machine learning library for Python designed to interoperate with the libraries NumPy[4] and SciPy[5]. It features various classification, regression and clustering algorithms such as SVM, decision trees, k-means clustering, gradient boosting etc. Today, for most people working on a data science project, scikit-learn is the most popular choice for access to easy-to-use and high-quality implementations of popular machine learning algorithms. In fact, it is the most popular and active open-source machine learning project in Python in terms of the number of contributors and commits[6].

Keras :
Keras is a Deep Learning library, written in Python, for Theano[7] and TensorFlow[8]. It is a high-level neural networks API capable of running on top of either TensorFlow or Theano. It attempts to enable the user to go from idea to result as quickly as possible by providing fast implementations of neural networks constructs and hence allows for fast prototyping. Keras supports both CNNs (convolutional neural networks) and RNNs (recurrent neural networks), as well as combinations of the two. The reason behind Keras' popularity is that it is a self-contained wrapper for deep learning i.e. one could use Keras to solve

problems end-to-end without ever having to interact with the underlying backend engine, Theano or TensorFlow.

We can observe that scikit-learn and Keras are quite popular machine learning libraries used for supervised learning tasks and thus are very good choices for integration with Gensim. Hence, the project would enable harmonious use of Gensim with the two libraries and would be of great use in industry as well as in academia.

A Task for Motivation : Movie Plot Classification

An interesting example of using Gensim for a supervised problem is the Movie Plot Classification Problem[9]. The scenario is as follows : *You run a movie studio. Every day you receive thousands of proposals for movies to make and you need to send them to the right department for consideration. There is exactly one department per genre so you need to classify plots by genre.*

As can be seen here, we can use Gensim in conjunction with scikit-learn to tackle the above-mentioned problem. Particularly, we have used Gensim's Word2Vec[10] and Doc2Vec[11] models with scikit-learn's classifiers for this particular instance of automated text-tagging problem.

More generally, we can use Gensim in conjunction with scikit-learn and Keras to address problems where we wish to utilise the output of any of Gensim's models further depending on the particular supervised task at hand. However, for any Gensim model to be directly compatible with scikit-learn elements such as pipeline.Pipeline and pipeline.FeatureUnions, it must provide functions like fit, transform, fit_transform etc. Similarly, for us to use any Gensim model with Keras, we need constructs like keras.layers.Embedding for making the models compatible with the Keras model. *This is exactly what this project aims to address.*

Implementation Details

Presently, the following models are present in Gensim :

1. **Author-Topic Model :** This model trains the author-topic model[12] on documents and corresponding author-document dictionaries. The training is online and is constant in memory with-respect-to the number of documents but is not constant in memory with-respect-to the number of authors. The model can be updated with additional documents after training has been completed. It is also possible to continue training on the existing data.
2. **Topic Coherence Model :** This model calculates topic coherence in python by implementing the four-stage topic coherence pipeline (Segmentation -> Probability Estimation -> Confirmation Measure -> Aggregation) based on [13]. Implementation of this pipeline allows for the user to in essence *make* a

coherence measure of one's choice by choosing a method in each of the pipelines.

3. **Doc2Vec Model :** This model enables us to learn representation for the input data through deep learning via the distributed memory and distributed bag of words models[14], using either hierarchical softmax or negative sampling.
4. **Hierarchical Dirichlet Process Model :** This model encapsulates functionality for the online Hierarchical Dirichlet Process algorithm[15]. It allows both model estimation from a training corpus and inference of topic distribution on new, unseen documents.
5. **Latent Dirichlet Allocation Model :** This model is based on the LDA technique for topic modeling[16] and allows both LDA model estimation from a training corpus and inference of topic distribution on new, unseen documents. The model can also be updated with new documents for online training.
6. **LogEntropy Model :** This model realizes the transformation between word-document co-occurrence matrix into a weighted matrix using the log entropy normalization, optionally normalizing the resulting documents to unit length.
7. **Latent Semantic Analysis Model :** This model is based on the LSA technique[17] and implements fast truncated SVD (Singular Value Decomposition)[18]. The SVD decomposition can be updated with new observations at any time for an online, incremental and memory-efficient training.
8. **Normalization Model :** This model realizes the explicit normalization of vectors and supports 'l1' and 'l2' norms.
9. **Random Projections Model :** This model allows building and maintaining a model for the Random Projections (or Random Indexing) technique[19].
10. **Term Frequency-Inverse Document Frequency Model :** This model is based on Tf-Idf weighting[20] and realizes the transformation between word-document co-occurrence matrix into a weighted tf-idf matrix.
11. **Word2Vec Model :** The model is based on the Word2Vec technique[21][22] and produces word vectors with deep learning via word2vec's skip-gram and CBOW models, using either hierarchical softmax or negative sampling.
12. **Dynamic Topic Model :** This model is based on the DTM technique[23] and implements topics that change over time and represents how individual documents predict that change.
13. **Text to Bag-of-words Model :** This model transforms input text into Gensim bag-of-words format corpus so that elements such as Pipeline could take text as input directly, rather than always having to take a bag-of-words corpus. The idea for a wrapper for this model has been proposed recently during the review of my PR #1244.

Further details about the above models can be seen from Gensim's API reference[24].

Example implementation for a wrapper for scikit-learn

As can be seen from API guidelines for scikit-learn objects[25], developing a wrapper for a Gensim model to work with scikit-learn would involve implementing the following functions of the new class :

1. **get_params** : This method takes no arguments and returns a dict of the __init__ parameters of the estimator, together with their values. It must take one keyword argument, deep, which receives a boolean value that determines whether the method should return the parameters of sub-estimators (for most estimators, this can be ignored). The default value for deep should be true.
2. **set_params** : This method takes as input a dict of the form 'parameter': value and sets the parameter of the estimator using this dict.
3. **fit** : Method to learn from data. Used like either estimator = obj.fit(data, targets) or estimator = obj.fit(data).
4. **transform** : Method to filter or modify the data, in a supervised or unsupervised way. Used as new_data = obj.transform(data).
5. **partial_fit** : Method to enable the model to learn incrementally. Used as estimator = obj.partial_fit(data).

On similar lines, I have submitted PR #1244 (**which has now been accepted and merged**) that creates a wrapper for the LSIModel. The code for the wrapper class has been explained through detailed comments below :

```
class SklearnWrapperLsiModel(models.LsiModel, TransformerMixin, BaseEstimator):
    """
    Base LSI module
    """


    def __init__(self, corpus=None, num_topics=200, id2word=None, chunksize=20000,
                 decay=1.0, onepass=True, power_iters=2, extra_samples=100):
        """
        Sklearn wrapper for LSI model. Class derived from gensim.model.LsiModel.
        """
        self.corpus = corpus
        self.num_topics = num_topics
        self.id2word = id2word
        self.chunksize = chunksize
        self.decay = decay
        self.onepass = onepass
        self.extra_samples = extra_samples
        self.power_iters = power_iters
```

```
        # if 'fit' function is not used, then 'corpus' is given in init
        if self.corpus:
            models.LsiModel.__init__(self, corpus=self.corpus, num_topics=self.num_topics,
              id2word=self.id2word, chunksize=self.chunksize, decay=self.decay,
              onepass=self.onepass, power_iters=self.power_iters,
              extra_samples=self.extra_samples)

    def get_params(self, deep=True):
        """
        Returns all parameters as dictionary.
        """
        return {"corpus": self.corpus, "num_topics": self.num_topics, "id2word": self.id2word,
                "chunksize": self.chunksize, "decay": self.decay, "onepass": self.onepass,
                "extra_samples": self.extra_samples, "power_iters": self.power_iters}

    def set_params(self, **parameters):
        """
        Set all parameters.
        """
        for parameter, value in parameters.items():
            self.parameter = value
        return self

    def fit(self, X,  y=None):
        """
        For fitting corpus into the class object.
        Calls gensim.model.LsiModel:
        >>>gensim.models.LsiModel(corpus=corpus, num_topics=num_topics,
            id2word=id2word, chunksize=chunksize, decay=decay,
            onepass=onepass, power_iters=power_iters, extra_samples=extra_samples)
        """
        if sparse.issparse(X):
            self.corpus = matutils.Sparse2Corpus(X)
        else:
            self.corpus = X

        models.LsiModel.__init__(self, corpus=self.corpus, num_topics=self.num_topics,
          id2word=self.id2word, chunksize=self.chunksize, decay=self.decay,
          onepass=self.onepass, power_iters=self.power_iters,
          extra_samples=self.extra_samples)

        return self

    def transform(self, docs):
        """
```

```
        Takes a list of documents as input ('docs').
        Returns a matrix of topic distribution for the given document bow, where a_ij
        indicates (topic_i, topic_probability_j).
        """
        # The input as array of array
        check = lambda x: [x] if isinstance(x[0], tuple) else x
        docs = check(docs)
        X = [[] for i in range(0,len(docs))];

        for k,v in enumerate(docs):
            doc_topics = self[v]
            probs_docs = list(map(lambda x: x[1], doc_topics))
            # Everything should be equal in length
            if len(probs_docs) != self.num_topics:
                probs_docs.extend([1e-12]*(self.num_topics - len(probs_docs)))
            X[k] = probs_docs
            probs_docs = []

        return np.reshape(np.array(X), (len(docs), self.num_topics))

    def partial_fit(self, X):
        """
        Train model over X.
        """
        if sparse.issparse(X):
            X = matutils.Sparse2Corpus(X)
        self.add_documents(corpus=X)
```

This class enables us to use Gensim models with scikit-learn elements such as pipeline.Pipeline and classifiers like linear_model.LogisticRegression as follows :

```
model = SklearnWrapperLsiModel(num_topics=2)

with open("/path/to/input/data/file",'rb') as f:
    compressed_content = f.read()
    uncompressed_content = codecs.decode(compressed_content, 'zlib_codec')
    cache = pickle.loads(uncompressed_content)

data = cache
id2word = Dictionary(map(lambda x : x.split(), data.data))
corpus = [id2word.doc2bow(i.split()) for i in data.data]

clf = linear_model.LogisticRegression(penalty='l2', C=0.1)

text_lda = Pipeline((('features', model,), ('classifier', clf)))
```

```
text_lda.fit(corpus, data.target)
score = text_lda.score(corpus, data.target)
```

Hence, creating wrappers for the remaining models would be on the same lines with the implementation for each function depending upon the particular model for which the wrapper is being created. For instance, for the Text to Bag-of-words Model, the fit function would learn a vocabulary dictionary for the input text, transform function would construct and return the bag-of-words model using the vocabulary created earlier and partial_fit function would update the vocabulary for the new text and the return the new bag-of-words model.

We should note that apart from the wrapper for LSI model mentioned above, Gensim already has a scikit-learn wrapper for LDA model[26]. So, we need to develop scikit-learn wrappers for the remaining 11 models.

Also, of the 11 models remaining to be implemented, Topic Coherence Model, LogEntropy Model, Random Projections Model, Dynamic Topic Model and Term Frequency-Inverse Document Frequency Model don't have an update function which allows the model to learn incrementally. Without such a function, we can't add the function partial_fit in the wrapper which is supposed to wrap the update function. So, we could either implement the update function or not implement the partial_fit for these models for now.

Example implementation for a wrapper for Keras

I have submitted PR #1248 (this work is in progress) which creates a wrapper for Word2Vec model in Gensim for integration with Keras.

```
from keras.layers import Embedding
from gensim import models


class KerasWrapperWord2VecModel(models.Word2Vec):
    """
    Class to integrate Keras with Gensim's Word2Vec model
    """


    def __init__(
            self, sentences=None, size=100, alpha=0.025, window=5, min_count=5,
            max_vocab_size=None, sample=1e-3, seed=1, workers=3, min_alpha=0.0001,
            sg=0, hs=0, negative=5, cbow_mean=1, hashfxn=hash, iter=5, null_word=0,
            trim_rule=None, sorted_vocab=1, batch_words=10000):

        """
        Keras wrapper for Word2Vec model. Class derived from gensim.model.Word2Vec.
        """
        self.sentences=sentences
        self.size=size
        self.alpha=alpha
```

```
        self.window=window
        self.min_count=min_count
        self.max_vocab_size=max_vocab_size
        self.sample=sample
        self.seed=seed
        self.workers=workers
        self.min_alpha=min_alpha
        self.sg=sg
        self.hs=hs
        self.negative=negative
        self.cbow_mean=cbow_mean
        self.hashfxn=hashfxn
        self.iter=iter
        self.null_word=null_word
        self.trim_rule=trim_rule
        self.sorted_vocab=sorted_vocab
        self.batch_words=batch_words

        models.Word2Vec.__init__(self, sentences=self.sentences, size=self.size,
            alpha=self.alpha, window=self.window, min_count=self.min_count,
            max_vocab_size=self.max_vocab_size, sample=self.sample, seed=self.seed,
            workers=self.workers, min_alpha=self.min_alpha, sg=self.sg, hs=self.hs,
            negative=self.negative, cbow_mean=self.cbow_mean, hashfxn=self.hashfxn,
            iter=self.iter, null_word=self.null_word, trim_rule=self.trim_rule,
            sorted_vocab=self.sorted_vocab, batch_words=self.batch_words)

    def get_embedding_layer(self):
        """
        Return a Keras 'Embedding' layer with weights set as our Word2Vec model's
        learned word embeddings
        """
        weights = self.wv.syn0
        layer = Embedding(input_dim=weights.shape[0], output_dim=weights.shape[1],
                    weights=[weights], trainable=False)
        return layer
```

The wrapper could be used in a Keras pipeline involving multiple layers using the Embedding layer returned by the function get_embedding_layer(). The code below shows one such instance where the wrapper has been used to address the 20NewsGroups problem[27].

```
from __future__ import print_function

import os
import sys
import numpy as np
```

```
from gensim.keras_integration.keras_wrapper_gensim_word2vec import KerasWrapperWord2VecModel
from gensim.models import word2vec
from keras.engine import Input
from keras.layers import merge
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils.np_utils import to_categorical
from keras.layers import Dense, Input, Flatten
from keras.layers import Conv1D, MaxPooling1D, Embedding
from keras.models import Model

BASE_DIR = ''
TEXT_DATA_DIR = BASE_DIR + './path/to/text/data/dir'
MAX_SEQUENCE_LENGTH = 1000
EMBEDDING_DIM = 100
VALIDATION_SPLIT = 0.2
BATCH_SIZE = 128

# prepare text samples and their labels

texts = []  # list of text samples
labels_index = {}  # dictionary mapping label name to numeric id
labels = []  # list of label ids

for name in sorted(os.listdir(TEXT_DATA_DIR)):
    path = os.path.join(TEXT_DATA_DIR, name)
    if os.path.isdir(path):
        label_id = len(labels_index)
        labels_index[name] = label_id
        for fname in sorted(os.listdir(path)):
            if fname.isdigit():
                fpath = os.path.join(path, fname)
                if sys.version_info < (3,):
                    f = open(fpath)
                else:
                    f = open(fpath, encoding='latin-1')
                t = f.read()
                i = t.find('\n\n')  # skip header
                if 0 < i:
                    t = t[i:]
                texts.append(t)
                f.close()
                labels.append(label_id)

# vectorize the text samples into a 2D integer tensor
```

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

word_index = tokenizer.word_index
data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)
labels = to_categorical(np.asarray(labels))

# split the data into a training set and a validation set
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
num_validation_samples = int(VALIDATION_SPLIT * data.shape[0])

x_train = data[:-num_validation_samples]
y_train = labels[:-num_validation_samples]
x_val = data[-num_validation_samples:]
y_val = labels[-num_validation_samples:]

# train the embedding matrix
data1 = word2vec.LineSentence('./path/to/input/data')
Keras_w2v = KerasWrapperWord2VecModel(data1, min_count=1)
embedding_layer = Keras_w2v.get_embedding_layer()

sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
embedded_sequences = embedding_layer(sequence_input)

x = Conv1D(128, 5, activation='relu')(embedded_sequences)
x = MaxPooling1D(5)(x)
x = Conv1D(128, 5, activation='relu')(x)
x = MaxPooling1D(5)(x)
x = Conv1D(128, 5, activation='relu')(x)
x = MaxPooling1D(35)(x)  # global max pooling
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
preds = Dense(len(labels_index), activation='softmax')(x)

model = Model(sequence_input, preds)
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['acc'])

model.fit(x_train, y_train, validation_data=(x_val, y_val), batch_size=BATCH_SIZE)
```

Hence, creating wrappers for the remaining models would be on the same lines with the implementation varying upon the particular model for which the wrapper is being created.

Related Work

This work would be a joint project with ShortText[28][29]. ShortText already has implementations for integrating Gensim with scikit-learn for Latent Dirichlet Allocation Model, Latent Semantic Analysis Model and Random Projections Model (see LatentTopicModeling and SkLearnClassification). ShortText also has wrappers for integration of various neural network algorithms in Keras (namely CNNWordEmbed, DoubleCNNWordEmbed, CLSTMWordEmbed and DenseWordEmbed) with Gensim where the input to the wrapper class is a pre-trained word-embedding model along with the class labels of the training data (see nnlib and sumvec).

Hence, we could take cues about the details of the implementation from ShortText wherever possible for the project.

Schedule of Deliverables

Note 1 : My summer vacation starts on 28th April, 2017. So I will be able to start coding one full month earlier than the GSoC coding period, effectively giving me 4 months to complete my project. Hence, I would be able to comfortably devote 50-60 hours a week for 4 months to work towards the proposed project.

Note2 : I would be writing tests for the wrappers that I develop along the way. This would assist me to spot and rectify bugs in my code quickly and also help me be confident about the correctness of the code that I write.

May 1st - May 28th, Community Bonding Period

- Before the official time period begins I intend to complete my ongoing work. This includes completing the PR for the Word2Vec wrapper for Keras (#1248). I have planned to complete this PR before the start of the community bonding period. This is because the problems that I could possibly face as well as the learnings that I would gain from completing this PR (as well as those from PR #1244) would help me to develop wrappers for other models as well. I would also like to wind-up PR #1201 which got put on the back-burner partly due to ongoing discussions about its implementation and partly due to my involvement with other PRs.
- In the past, I have used scikit-learn, TensorFlow and Theano often in my coursework as well as for my pet projects. This has enabled me to be fairly well-versed with the APIs and usage of these libraries. However, I haven't used Keras directly in any project so far. So, I would like to get my hands dirty working with Keras by implementing some small programs involving

neural networks. Since Keras is built on top of TensorFlow and Theano, this should not pose a problem for me.
- I would also actually like to get started with the coding of the wrappers within this period. This would not just give me extra time to complete the work but also get fully versed with Gensim's codebase. I would also try to help to fix some of the existing (and identify new) bugs in code as well as errors in documentation that I come across here. Thus, this phase would involve triage of new and existing issues as well.
- I have been fairly active on Gensim's Gitter channel as well as the mailing list, resolving queries of other members whenever I could. The community too has been very forthcoming right from the start and on several occasions I have been guided in the right direction by the members. Thus, I plan to do my bit by participating on both these mediums during the entire summer.
- A big chunk of open-source development is sharing your ideas and work with others. For this, I've started a blog here to report the work that I would be doing on a weekly basis.

May 29th - June 3rd

- Start developing wrappers for the models mentioned here for scikit-learn.

June 5th - June 9th

- Continue wrapper implementation for integration with scikit-learn.
- Wrappers for half of the models should be implemented by now.

June 12th - June 16th

- Continue wrapper implementation for integration with scikit-learn.

June 19th - June 23th, End of Phase 1

- Complete wrapper development for integration with scikit-learn.
- The writing of wrappers for integration with scikit-learn is expected to be completed by the end of this week. However, if this is not the case, some part of the following week can be devoted to finish the remaining work.

June 26 - June 30th, Begin of Phase 2

- Finish incomplete wrappers for scikit-learn, if any.
- Start developing wrappers for the models mentioned here for Keras.
- Regularly update the blog as well.

July 3rd - July 7th

- Continue wrapper implementation for integration with Keras.

July 10th - July 14th

- Continue wrapper implementation for integration with Keras.
- Wrappers for half of the models should be implemented by now.

July 17th - July 21th, End of Phase 2

- Complete wrapper development for integration with Keras.
- The writing of wrappers for integration with Keras is expected to be completed by the end of this week. However, if this is not the case, some part of the following week can be devoted to finish the remaining work.

July 24th - July 28th, Begin of Phase 3

- Finish incomplete wrappers for Keras, if any.
- Start creating ipython notebook tutorials for each wrapper to make it easier for others to utilise them. Some of the use-cases implemented in IPython tutorials are expected to overlap with the unit-tests created earlier, so we could use our earlier code and learnings here.

July 31st - August 4th

- Continue working on IPython tutorial notebooks.

August 7th - August 11th

- Complete ipython notebook tutorials for all wrappers.

August 14th - August 18th

- Finish incomplete IPython tutorials, if any.
- Rectify possible bugs and problems our implementation would have at this point.
- Ensure all edge and corner cases have been handled appropriately.
- Optimize the code on the fronts of time, memory usage and accuracy for improved efficiency.

August 21st - August 25th, Final Week

- Document all useful information regarding aspects such as parameter selection and model tuning from the insights gained so far.
- Optimize and fine-tune the code further wherever possible.
- Scrub and wrap-up documentation.

August 28th - August 29th, Submit final work

- Get the pull request merged.
- Write concluding blog post.

Future works

Our primary reasons for attempting to integrate Gensim with scikit-learn and Keras are ease-of-use, popularity and the wide range of usage of these libraries. On similar grounds, we could integrate Gensim with spaCy[30] as well so that the tasks which spaCy excels at, such as data pre-processing, could be done seamlessly while working with Gensim. textacy[31] is one such library that provides wrappers around spaCy's various functionalities for tasks such as text pre-processing, information extraction, topic modeling (using scikit-learn) etc. Hence, we could take some idea from textacy's code about implementation details for this work.

Apart from this, I plan to stick around and help others to resolve any issues related to this project which may emerge in the future.

Development Experience

Currently I am working with a tech startup called PregBuddy. So far, I have developed a Facebook messenger bot which was shipped in December 2016 and has helped us increase our number of users significantly. Also, I worked on a Google Home bot using api.ai which has not been released to users till now. Currently, I am developing an in-app chatbot for our Android application which is expected to be shipped by the end of April, 2017.

During the summers of 2016, I was an intern at Nomura, Mumbai where I worked with their Quantitative Research and Information Technology teams to develop a system to automate the process of running very large scale risk-value computations and comparisons. These computations were supposed to be run everyday after the closing of stock markets of various countries monitored by Nomura. The system was completed and released for use during the last week of my internship period across all offices of Nomura globally. The internship not just gave me an experience of developing software while working in a large team but also enabled me to refine the skill of jumping into and working with large and unfamiliar codebase.

Both these experiences have helped immensely me to gain real-life understanding about what really goes into developing products for people on a large scale.

Other Experiences

The most important reason behind my desire to contribute to Gensim has been my previous experience with Natural Language Processing and Machine Learning. As a part of my coursework, I have undertaken a course on *Machine Learning*, which had a lab component as well. The project which I worked on was *Anomaly Detection in Surveillance Videos* in which we developed a system to detect anomalies in surveillance video-feeds for pedestrian walkways. Various machine learning approaches and techniques like Optical Flow and AlexNet for the task of feature extraction and Decision Trees, Support Vector Machine, K-Nearest Neighbors, Naïve Bayes, among others, for classification were used. In addition, Time Series Analysis and Topic Modelling approaches were also implemented and tested.

As a sophomore, I was an intern in the Language Technologies Research Center at International Institute of Information Technology Hyderabad where I worked on the project *Question Answering Systems using Deep Learning* which aimed at solving the problem of open-domain single-relation question-answering. I have also worked on a semester-long project on *Citation Recommendation using Citation Context* which aimed to solve the problem of recommending citations to the author using different approaches for document similarity. In addition to this, we tested and applied various methods for spell correction and query expansion (including Gensim's similar_by_word method) as well.

Right now, I am undertaking a course on *Deep Learning* which has enabled me to get a solid understanding of the recent research developments in the field and also given me an opportunity to implement some of these developments myself. I like to work with *not-too-big* and interesting problems whenever I get time by implementing and tinkering around their solutions in the form of pet-projects. Some of them are :

- Word-derivation task
- MNIST using CNN
- Word-analogy
- MNIST through multilayer perceptron using TensorFlow
- MNIST through multilayer perceptron without neural network libraries
- Word similarity using Glove
- Search algorithms for 8-puzzle problem

For getting a concise and complete view about my previous work, you can go through my resume.

Hence, the above experiences have given me a strong understanding of the concepts and fundamentals associated with the fields of NLP and Machine

Learning, in addition to a hands-on experience to write code to address a real task at hand.

Contributions

As of now, I have submitted 8 pull requests to Gensim and 1 to ShortText. Doing this has helped me immensely in getting comfortable with the codebase and also learning about the community's coding standards and practices.

1. Added scikit-learn wrapper for LSI model in Gensim (#1244) : This PR also involved adding unit-tests and updating the IPython notebook for the wrapper.

2. Added function `predict_output_word` to predict the output word given the context words (#1209) : This fixed issue #863.

3. Updated Word2Vec's IPython notebook for the function `predict_output_word` (#1228)

4. Added Keras wrapper for Word2Vec model in Gensim (#1248) : This work is in progress currently as an IPython notebook needs to be created for the wrapper. I plan to finish this PR very soon in the next couple of days.

5. Allows training if model is not modified by `_minimize_model` (#1207) : This PR was based on the bug in the code that I found for the function `_minimize_model` (relevant Google mailing list discussion). This PR also adds a deprecation warning for this function.

6. Computes training loss for skip gram model (#1201) : This PR computes the loss while training a Word2Vec model. This PR intends to fix issue #999. This work is in progress as it got deferred temporarily due to the ongoing discussions about the implementation as well as my involvement with other PRs.

7. Updated description for `worker_loop` function used in `score` function (#1206)

8. Fixed duplicate imports and typos in word2vec.py (#1240)

9. Fixed typographical errors in ShortText's documentation (#2)

Why this project?

I have frequently used scikit-learn, TensorFlow and Theano in my projects, both within and outside my coursework. Since this project aims to integrate Gensim with these popular libraries, this project can be expected to help a lot of people in

the industry and academia who prefer using Gensim and are trying to solve a real-life supervised task. Moreover, I plan to work in the domain of topic modelling for my M.Tech. thesis as well.

Hence, seeing the potential impact of the project and its overlap with my interest and previous work, the project piqued my interest.

Appendix

Below is a list of all the web articles, libraries, code snippets and research papers mentioned in the proposal.

1 - http://radimrehurek.com/gensim/
2 - http://scikit-learn.org/
3 - https://keras.io
4 - https://www.numpy.org/
5 - https://www.scipy.org/
6 - http://www.kdnuggets.com/2015/06/top-20-python-machine-learning-open-source-projects.html
7 - https://github.com/Theano/Theano
8 - https://www.tensorflow.org/
9 - https://github.com/RaRe-Technologies/movie-plots-by-genre
10 - https://radimrehurek.com/gensim/models/word2vec.html
11 - https://radimrehurek.com/gensim/models/doc2vec.html
12 - https://mimno.infosci.cornell.edu/info6150/readings/398.pdf
13 - http://svn.aksw.org/papers/2015/WSDM_Topic_Evaluation/public.pdf
14 - https://arxiv.org/pdf/1405.4053v2.pdf
15 - https://en.wikipedia.org/wiki/Hierarchical_Dirichlet_process
16 - https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation
17 - https://en.wikipedia.org/wiki/Latent_semantic_analysis
18 - https://en.wikipedia.org/wiki/Singular_value_decomposition
19 - https://en.wikipedia.org/wiki/Random_projection
20 - https://en.wikipedia.org/wiki/Tf%E2%80%93idf
21 - https://arxiv.org/abs/1301.3781
22 - https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf
23 - https://en.wikipedia.org/wiki/Dynamic_topic_model
24 - http://radimrehurek.com/gensim/apiref.html
25 - http://scikit-learn.org/stable/developers/contributing.html#apis-of-scikit-learn-objects
26 - https://github.com/RaRe-Technologies/gensim/pull/932
27 - http://qwone.com/~jason/20Newsgroups/
28 - https://pythonhosted.org/shorttext/
29 - http://gibbslda.sourceforge.net/fp224-phan.pdf
30 - https://spacy.io/
31 - http://textacy.readthedocs.io/en/latest/