Knowledge Base Embeddings for DBpedia

Personal Information

- Contact Information
  - Name: Akshay Jagatap
  - GitHub User ID: @akshayjagatap
  - Email: …
  - Phone: …
  - Location: …

Project

- Project Description:

  The project aims at defining embeddings to represent classes, instances and properties. Such a model tries to quantify semantic similarity as a measure of distance in the vector space of the embeddings. Semantic distance is not just a measure of similarity in meaning, but also a measure of relatedness. For example, “car” and “truck” are similar because both are modes of transportation using a road as a medium which move using wheels driven by an engine, both belong to the class of “Road Vehicles”. On the other hand, “car” and “road” don’t appear to be similar, cars are a mode of transportation, while roads are a medium through which cars transport, but are indeed related, because they both belong to the category of “Transportation”, and both can be defined to have

“transportation based applications” in their definitions, and are hence are similar in that part of their definition. Semantic distance between two objects can rather be thought of as a measure of descriptiveness of the common class which the two objects belong to. If two objects have a very low semantic distance, they belong to a common class with a very high level of description, as the distance increases the common class drops in the level of description and complexity. For example, for all instances, :Thing is a common class, but it doesn’t really define anything about the instances, on the other hand, for any two people, :Person is the common class and the description of human beings apply to them.

We are trying to better represent knowledge to refine the DBpedia KB, which when coupled with an inference engine can deduce new knowledge. Wikipedia is the largest reference work on the internet, holding a comprehensive summary of information from all branches of knowledge. Generating a thesaurus like vector space using a KB derived from such a huge corpus should improve entity linking, relationship extraction, information retrieval, concept mining and machine translation. We are trying to derive some sort of conceptual knowledge from DBpedia graphs, which will ease machine-access of this information. In order to evaluate the integrity of the derived embeddings, we will mainly be using the Entity Linking Evaluation, to assess how well an entity can be derived from its context and the Analogical Reasoning Task, which favors models that produce dimensions of meaning.

- ## Basic Ideology

I have gone through the papers linked on the ideas page of Knowledge base embeddings. I have read the following papers as well.

1. TransG : A Generative Model for Knowledge Graph Embedding
2. An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors
3. Calculating Semantic Distance between Word Sense Probability Distributions
4. Scalable Techniques for Creating Semantic Vector Representations
5. Parameterized context windows in Random Indexing
6. PSDVec: a Toolbox for Incremental and Scalable Word Embedding
7. Learning to Link with Wikipedia

From these papers I believe “Scalable Techniques for Creating Semantic Vector Representations” points the way forward in defining the semantic vectors to represent

DBpedia resources, which is Random Vector Accumulators (RVA). The algorithm begins in a manner similar to Random Indexing by mapping every piece of information to an initial random hyper-dimensional vector. The components (generated using Gaussian distribution with $\mu = 0$ and $\sigma = 1/\sqrt{n}$, where n is the dimension of the vector) of these vectors are a small number of positive and negative values and a large number of zeroes. Owing to their hyper-dimensionality and having a large number of zeroes these vectors are nearly orthogonal. The ease with which orthogonal vectors are generated is one of the reasons I propose using this model. When a text corpus is processed, a Word's lexical memory vector is set as the sum of these random vectors corresponding to the words appearing in context window of the Word.

Now, the cosine between the lexical memory vectors of any two words represents a measure of their relatedness. We define this lexical memory vector to be the embedding of the Word. And from what I understood reading "Scalable Techniques for Creating Semantic Vector Representations" is that the RVA scales impressively to large datasets and are efficient to compute on a laptop and for a colossal dataset like DBpedia, I believe RVA should outperform Latent Semantic Analysis, because while scaling up to large datasets it is tougher to perform SVD and the memory requirements of LSA increase exponentially, on the other hand RVAs capitalize on additional information rather than being hindered by it. Additionally, inclusion of new data is easier in RVAs as compared with LSA. I intend to work on adding a few modifications to the RVA implementation to achieve better results.

- Evaluation

  - **Entity Linking [Primary Evaluation]**
    Given a paragraph with a link, L anchored to text, A we try to define an embedding for A defined using our implementation. The K closest vectors to the embedding of A should contain the embedding of link L for the evaluation to be marked as successful. Giving this paragraph as an input to DBpedia Spotlight, it is possible to get N nearest predictions for L, we use these results to determine a baseline for the accuracy of our implementation. Datasets from Open Knowledge Extraction (OKE) Challenge or other entity recognition challenges can be used the datasets can be found here. Wikipedia data held out of DBpedia can be used

as well. The OKE Challenge dataset linked above have root entities distributed in the following manner:

Class Avg.	instances per page	Avg. properties per page	# uniq. Hosts
MusicRecording	2.52	11.77	154
Person	1.56	7.71	2970
Recipe	1.76	21.95	2387
Restaurant	3.15	14.69	1786
SportsEvent	4	14.28	135
Mixed	2.26	14.42	7398

- **Analogical Reasoning Task [Secondary Evaluation]**
  Our embeddings should be able to semantic analogies such as
  India : New Delhi = Canada : ?,
  to find the answer that matches the analogy we try to find the vector closest to $V_{\text{New Delhi}} - V_{\text{India}} + V_{\text{Canada}}$. Such problems have long been asked in the SAT Analogies and Comparisons Section, the datasets I found are as follows:
  1. Google Analogy Test Set: has 19,544 questions of which over 4,435 are state : capital relationships.
  2. Bigger Analogy Test Set: has 99,200 questions, which includes an encyclopedic semantics section having name : occupation, male : female, country : capital, animal : shelter, things : colour, name : nationality, country : language relationships.
  3. SemEval 2012 Task 2: has 3,218 questions which include space : time, class : inclusion, part : whole, cause : purpose relationships to name a few.

❖ **Additional Evaluative Components**

- **Triad Comparison**
  Additionally, a triad based system can be used for evaluation, given T-A-B, where T is the target term, the algorithm needs to select one of A and B such that the chosen one is closer in semantic distance than the other. The dataset is defined by choosing two embeddings from the same class and a third embedding from a

different one. The algorithm should be able to determine which two embeddings belong to the same class. For e.g we select :Cat, :Lion and :Kiwi at random, our knowledge base should be able to determine that :Cat and :Lion are semantically closer than either of them with a :Kiwi.

- **Odd One Out**

  In odd-one-out problems, we are given a set of options from which one option is distinct from the others. All the odd-one-out/odd-man-out problems depend on an estimation of the most semantically distant option. Though no use-as datasets exist, such problems have been well documented with solutions to these problems (and have long been a part of standardized tests such as GRE), after some textual cleaning such a dataset can be generated. Given that our code can measure semantic distance between the options and classify the most distant one as the “odd-man-out”.

- **Class Recognition**

  To check how well similar classes are clustered together, we give an input vector v and find its k nearest vectors and return their most common class (i.e the class that all the k instances belong to) and check if the instance in question belongs to that class, i.e given that an instance ‘A’ belongs to some class ‘C’, then the k nearest vectors to ‘A’ being V_{A1}, V_{A2}, V_{A3}….V_{Ak}. And the class that embodies all these vectors with the highest level of description be C_{all} then C should be a subclass of C_{all}.

● Implementation

In order to utilize the intricacies of the Wikipedia corpus, the Random Vector Accumulator needs to be implemented in such a manner. Firstly, we would like to use information from DBpedia graphs to help generate the embeddings. To generate the embeddings we simply add the random vectors corresponding to the property values of the DBpedia entity with its lexical memory vector. For e.g. the DBpedia page for :The_Beatles has :Rock_music and :Pop_Music listed as the value of the :genre property. Thus while generating the embeddings for :The_Beatles we simply add the random vectors of :Pop_Music and :Rock_music to the lexical memory of :The_Beatles.

Secondly, utilizing wikipedia links and anchor texts; firstly, on a Wikipedia Article, anchor texts are words significant enough to have a Wikipedia article of their own, so these mark the words that define the article and are important to the definition of the Article and hence should be given a higher weightage in the lexical memory of the Article than regular words appearing in it. By multiplying the random vectors of page links with a hyperparameter weight, this hyperparameter is tuned using a random search.
Outcome: Given “The constellation's most obvious deep-sky object is the naked-eye Andromeda Galaxy.” in the abstract of :Andromeda. While summing the random vectors with the lexical memory, “Andromeda Galaxy” will be given a higher weight as compared to the words “deep-sky” or “naked-eye”. Thus the lexical memory vector will have a greater magnitude in the direction of the random vector of Andromeda_Galaxy.

Thirdly, Wikipedia links are anchored to texts that represent the class which the instance belongs to with respect to the context of discussion, these context based class names also need to be given a weightage in the lexical memory of the article it directs to. These class names need not be actual DBpedia classes, but rather classes in a more general sense, for example in Boost (C++ libraries), the link to IEC defined ISO standard for C++ is linked to anchor text C++ Standards. So these context based anchor texts used to define the entities need to be given a greater weightage in the embedding of the entity.
Outcome: The link to Barack Obama can be anchored to President of the United States or The Most Powerful Person in the World, i.e the class which the instance Barack Obama belongs to, such context based class names need to be included in the lexical memory after multiplication with a weight factor in a manner similar to the previous.

Since some properties define a word better than others, we need to assign weights to these properties on the basis of their importance in describing the word. For example, knowing that The Beatles is a Band is more important than knowing that they are a Thing. Thus, while defining the lexical memory of :The_Beatles, rather than simply summing the random vectors of :Band and :Thing, we would want to give a greater weightage to the :Band vector. This can be done in a manner similar to tf-idf. We define a class frequency (the number of instances that belong to a particular class) and an importance measure such that a high frequency gives a low importance measure and a low frequency gives a high importance measure. Now we use this importance measure to determine the weight needed to be given to random vector of the class before summing it with the lexical memory of the resource it represents. Similarly we can

encode DBpedia properties in the embeddings by taking a weighted sum of the random vectors of these properties with the lexical memory vector.
Outcome: :Band will have fewer number of instances as compared to :Thing and thus while generating the lexical memory of :The_Beatles, the random vector of :Band will be multiplied by a greater factor as compared with :Thing. Thus the embedding for :The_Beatles has a greater magnitude in the direction of the random vector of :Band than the random vector of :Thing.
Risk Involved: This feature need not give positive results as it is subject to how well the classes have been defined for the instances, which need not always be the case.

Rather than depending on the class definition of an instance in order to improve the embedding, I would like to define a measure of relatedness between two Wikipedia articles by comparing their incoming and outgoing links, as described in Learning to Link with Wiki. Using this measure we can avoid using hyperparameters and better encode semantic information in the embeddings by scaling our random vectors using this relatedness measure before addition with the lexical memory of the vector.
Outcome: We update the lexical memory vector of say :The_Beatles in the following manner:

$$V(:The\_Beatles) \mathrel{+}= (1 + Relatedness(:The\_Beatles, :Band)) * V_{Random}(:Band)$$

We already have code capable of measuring the similarity between two wikipedia articles. Now we need to encode this information in the vector space of the embeddings such that the distance between any two embeddings gives a measure of their relatedness. The RVA has allowed us to encode semantic information of different entities in a hyperdimensional vector. We can perform convolutions on this vector and further pass it into a neural network to obtain a new embedding. This neural network is updated with stochastic gradient descent updates which try to fit the distances between embeddings with their relatedness measure.
Outcome: The obtained embeddings better embody the relatedness measure as the distance between the new embedding.

This is but a rough overview I have on the implementation, which I have tried to structure in the project timeline.

- ## Project Timeline
    - **Community Bonding period (Till 30th May)**
      [Acquire Datasets] [First Evaluation]
      This will be a great time to get to know everybody and I'm very excited to learn from them. The remaining time would go into reading the documentation and getting to know the practices followed in the community. I will also use this time to read the documentation for the tools I will be using, so that I know how to use them properly and am aware of the best way to achieve the result and make informed decisions. My summer vacation starts May 17th. This period of time will mostly involve acquiring and generating the datasets and determining an initial baseline. We generate the baseline by using existing embeddings such as wiki2vec, GloVe and Dependency-Based Word Embeddings which are derived from Wikipedia.

    - **Week 1 and 2 (30th May - 11th June)**
      [Generating embeddings using existing software]
      Firstly, before implementing any new ideas, we would like to maximally use existing software for our purpose. Most importantly the software needs to be scalable to the size of Wikipedia and DBpedia. I'll be using PSDVec, a low matrix factorization based method, which uses eigendecomposition, which avoids the information loss in SVD based methods and results in higher quality embeddings. PSDVec allows incremental learning of new words; after defining the general vocabulary and incrementally learn new words which appear rarely. Secondly I'll be using the Random Indexing implementation of S-Space. In order to learn the embeddings, we will initially be using the Wikipedia dump. Using regex, the wikicode can be converted to text retaining both the anchor text and the linked entity. For example the following wikicode from Trees,
      "Plants such as [[Arecaceae|palms]], bananas and papayas are not considered trees." which is refactored into,
      "Plants such as :Arecaceae palms, bananas and papayas are not considered trees."
      Additionally, "With members [[John Lennon]]...." will be refactored to
      "With members :John_Lennon John Lennon" resulting in :John_Lenon to have a high correlation with both the words John and Lennon. Thus multi-word titles can be handled in this manner. Following this, rest of the xml code can be cleared off using the inbuilt xml library in python. This "clean" corpus when fed to S-Space and PSDVec will generate embeddings for the words including the DBpedia entities.

- **Week 3 and 4 (12th June - 25th June) [Scaling up][Second Evaluation]**
  Scale up by giving wikipedia dump as the input after cleaning out the xml code from it. Carry out the evaluation methods.

- **Week 4 and 5 (26th June - 9th July) [RVA Implementation]**
  Begin implementing the RVA including the features as discussed in the implementation section. Initially we work with the weights set at one, giving all features equal weightage. The weights will be determined in the evaluation section depending on the accuracy they offer on the evaluation procedures. Also parametrize the context windows such that the random vectors of the context words are scaled both to their relative frequency and usefulness. This can be done by defining weights that are penalized for a greater occurrence frequency and the parameter of usefulness is optimized through back-propagation such that resulting embeddings fit the relatedness measure as distances between embeddings in the hyper-dimensional vector space. Thus we try to define,

  $$V = \theta.v$$

  Where V is the lexical memory vector, θ is a column vector that holds the weights to be assigned to its corresponding random vector. v is a matrix such that its columns are the random vectors of the context words and the number of columns in the matrix is equal to the context window size.

- **Week 6 and 7 (10th July - 23 July)**
  [Hyperparameter Tuning][Third Evaluation]
  Determine the hyperparameters, the weight needed to be assigned to page links, dbpedia entities and the anchor text. Carry out the evaluation to get a baseline for the RVA.

- **Week 8 and 9 (24th July - 6th August) [Improve Embeddings]**
  Since the word embeddings are generated using co-occurrence information from a large corpus, it is likely that the obtained embeddings will be very general and reduce the expressiveness of the embedding. We use the relatedness measure we defined earlier, between articles which we use to scale the random vectors before summing them into the lexical memory. This way we can better encode the semantic information of entities in their embeddings, and even replace the hyperparameters with this relatedness measure.

- **Week 10 and 11 (7th August - 20th August)**
  [Improve Embeddings][using CNN]
  Using our RVA we have managed to encode the resource (based on its context) into an embedding, to better these embeddings, (i.e. we want to push similar resources closer and pull dissimilar resources apart in the vector space). We can now learn features that make the embeddings related in a semantic sense by define a CNN, that performs convolutions on lexical memory vector, and further apply stochastic gradient descent updates on the neural network by using a cost function that tries to fit the distance between embeddings to the relatedness measure we had defined earlier; and obtain an embedding that can give a better measure of semantic distance than the original embedding. Thus we try to minimize the weights of the neural network so as to fit the distance in order to minimize the following cost function,

  $$\Sigma(\text{distance}(V_i,V_j) - \text{relatedness}(V_i, V_j))^2$$

  As “Parameterized context windows in Random Indexing” suggest, applying SGD Random Indexing gives better results than vanilla RI for the purpose of determining word similarity.

- **Week 12 and 13 (21st August - 3rd September)**
  [Final Evaluation][Cleaning up the code]
  Implementing fixes and perform a final testing of the model. And finally will mostly involve cleaning up the code and refactoring, fixing bugs and documentation.

  Note: I will also be blogging weekly about my progress during the summer. The timeline might seem crude as at this moment I am not able to judge how much time each work is going to take. I will proceed to the next part if something finishes earlier. Also there is the buffer period if a bug delays development.

Technical Skills

I am comfortable using

- **C/C++**: It's the first language I learnt, have been using it for 6 years. I code in C++ mainly for competitive programming challenges.
- **Python**: Have been using it for 1.5 years. I use python to solve machine learning and data analytics challenges.
- **Java**: Learnt Java a semester ago as a part of my OOP course.
- **Bash**: Have been using it for a year, since I dual booted with Debian. I use it for developing handy code fragments to perform repetitive tasks.

Along with these languages I have basic knowledge of Scala, Tensorflow and Theano. I have also worked with deep learning networks using Caffe for object classification in images.

Open Source

This is my first attempt at open source development. What intrigues me the most is that something as awesome as Linux, Firefox and VLC could be created by a community driven approach. I have long been using open source software like VLC and Firefox and a few years ago I started using Debian. I believe it is time I gave back to the open source community and now is a time better than any.

Background & Education

I am in my third year of pursuing a Masters in Physics and a Bachelor of Engineering in Computer Science at Birla Institute of Technology and Science, K. K. Birla Goa Campus, Goa, India, mine is a dual-degree five year course and I'm expected to graduate in October, 2019.

Summer Plans

- **What city/country will you be spending this summer in?**

  …

- **Do you have a full- or part-time job or internship planned for this summer?**

  …

- **How many hours per week do have available for a summer project?**

  …

GSoC Experience

- **Did you participate in a previous Summer of Code project?**
  …

- **Have you applied or do you plan to apply for any other 2017 Summer of Code projects?**
  …

- **Why did you decide to apply for a DBpedia Spotlight project?**
  …

If required by the mentors, I will make the appropriate changes to my proposal post submission. I really want to see this project happen over the course of the summer. I have basic knowledge in the field of NLP, I have been trying to improve upon that by reading papers and even completing Hugo Larochelle's Deep Learning course on NLP, I am in the process of completing CS224n: Natural Language Processing with Deep Learning.

And as for why I should be taken, I have experience working with neural networks involving image recognition, and since image processing methodologies are being improvised to handle the high dimensionality of the vector spaces involving word embeddings, I believe I will have a better understanding of how to deal with the task at hand.