Introduction to Recommender Systems (from the perspective of a numerical analyst)

Jack Poulson (Hodge Star Scientific Computing)
Aussois, France, June 20, 2019

From video sharing websites, to NLP, to search engines, to online stores, one of the most fundamental questions to ask is: which items are most similar to a given one?

How do we define similarity?

- We could determine locations on a manifold (e.g., Euclidean space, a sphere, or hyperbolic space) for each item and define similarity based upon their distance.
- Practitioners call this an **embedding**, but be aware that differential topologists/geometers will object: there is no guarantee of even injectivity, and, in fact, it is typically lost ("**folding**").
- But surely there are no good low-dimensional representations of complex ideas?
- Millions of pages of analysis have been written based upon the 1D embedding of political opinion: "left-wing", "left of center", "right-wing", etc...
- Typical word/term embeddings (e.g., word2vec, GloVe, fastText) use a 200-500 dimensional sphere with cosine distance.

How do we define similarity?

- We could determine locations on a manifold (e.g., Euclidean space, a sphere, or hyperbolic space) for each item and define similarity based upon their distance.
- Practitioners call this an **embedding**, but be aware that differential topologists/geometers will object: there is no guarantee of even injectivity, and, in fact, it is typically lost ("**folding**").
- But surely there are no good low-dimensional representations of complex ideas?
- Millions of pages of analysis have been written based upon the 1D embedding of political opinion: "left-wing", "left of center", "right-wing", etc...
- Typical word/term embeddings (e.g., word2vec, GloVe, fastText) use a 200-500 dimensional sphere with cosine distance.

How do we define similarity?

- We could determine locations on a manifold (e.g., Euclidean space, a sphere, or hyperbolic space) for each item and define similarity based upon their distance.
- Practitioners call this an **embedding**, but be aware that differential topologists/geometers will object: there is no guarantee of even injectivity, and, in fact, it is typically lost ("**folding**").
- But surely there are no good low-dimensional representations of complex ideas?
- Millions of pages of analysis have been written based upon the 1D embedding of political opinion: "left-wing", "left of center", "right-wing", etc...
- Typical word/term embeddings (e.g., word2vec, GloVe, fastText) use a 200-500 dimensional sphere with cosine distance.

How do we define similarity?

- We could determine locations on a manifold (e.g., Euclidean space, a sphere, or hyperbolic space) for each item and define similarity based upon their distance.
- Practitioners call this an **embedding**, but be aware that differential topologists/geometers will object: there is no guarantee of even injectivity, and, in fact, it is typically lost ("**folding**").
- But surely there are no good low-dimensional representations of complex ideas?
- Millions of pages of analysis have been written based upon the 1D embedding of political opinion: "left-wing", "left of center", "right-wing", etc...
- Typical word/term embeddings (e.g., word2vec, GloVe, fastText) use a 200-500 dimensional sphere with cosine distance.

How do we define similarity?

- We could determine locations on a manifold (e.g., Euclidean space, a sphere, or hyperbolic space) for each item and define similarity based upon their distance.
- Practitioners call this an **embedding**, but be aware that differential topologists/geometers will object: there is no guarantee of even injectivity, and, in fact, it is typically lost ("**folding**").
- But surely there are no good low-dimensional representations of complex ideas?
- Millions of pages of analysis have been written based upon the 1D embedding of political opinion: "left-wing", "left of center", "right-wing", etc...
- Typical word/term embeddings (e.g., word2vec, GloVe, fastText) use a 200-500 dimensional sphere with cosine distance.

How do we compute embeddings of items?

- Goal is to map each item – word of English, user of website, product in store, ... – (roughly) onto an $(r-1)$-sphere ($r \approx 200$).
- There is an **artform** to determining the model, but, roughly speaking, one finds a low-rank approximation of a very large (possibly implicit) sparse matrix.[1]
- For **word embeddings**, entry (i,j) might be $\log(c_{i,j}+1)$, where $c_{i,j}$ is the weighted sum of coocurrences of word i with word j in Wikipedia (perhaps with k words of separation leading to a contribution of $1/k$).
- For **video embeddings**, entry (i,j) might be the **squashed** count of the number of times video i was watched after video j or by user j.

[1] In the nonlinear context, one speaks of building an **autoencoder**.

How do we compute embeddings of items?

- Goal is to map each item – word of English, user of website, product in store, ... – (roughly) onto an $(r-1)$-sphere ($r \approx 200$).
- There is an **artform** to determining the model, but, roughly speaking, one finds a low-rank approximation of a very large (possibly implicit) sparse matrix.[1]
- For **word embeddings**, entry (i,j) might be $\log(c_{i,j}+1)$, where $c_{i,j}$ is the weighted sum of coocurrences of word i with word j in Wikipedia (perhaps with k words of separation leading to a contribution of $1/k$).
- For **video embeddings**, entry (i,j) might be the **squashed** count of the number of times video i was watched after video j or by user j.

[1] In the nonlinear context, one speaks of building an **autoencoder**.

How do we compute embeddings of items?

- Goal is to map each item – word of English, user of website, product in store, ... – (roughly) onto an $(r-1)$-sphere ($r \approx 200$).
- There is an **artform** to determining the model, but, roughly speaking, one finds a low-rank approximation of a very large (possibly implicit) sparse matrix.[1]
- For **word embeddings**, entry (i,j) might be $\log(c_{i,j}+1)$, where $c_{i,j}$ is the weighted sum of coocurrences of word i with word j in Wikipedia (perhaps with k words of separation leading to a contribution of $1/k$).
- For **video embeddings**, entry (i,j) might be the **squashed** count of the number of times video i was watched after video j or by user j.

[1] In the nonlinear context, one speaks of building an **autoencoder**.

How do we compute embeddings of items?

- Goal is to map each item – word of English, user of website, product in store, ... – (roughly) onto an $(r-1)$-sphere ($r \approx 200$).
- There is an **artform** to determining the model, but, roughly speaking, one finds a low-rank approximation of a very large (possibly implicit) sparse matrix.[1]
- For **word embeddings**, entry (i, j) might be $\log(c_{i,j} + 1)$, where $c_{i,j}$ is the weighted sum of coocurrences of word i with word j in Wikipedia (perhaps with k words of separation leading to a contribution of $1/k$).
- For **video embeddings**, entry (i, j) might be the **squashed** count of the number of times video i was watched after video j or by user j.

[1] In the nonlinear context, one speaks of building an **autoencoder**.

Nearest neighbors from GloVe embeddings

`https://nlp.stanford.edu/projects/glove/`

- The typical training set is Wikipedia (historically, also non-free Gigaword 5).
- Main example from website is nearest neighbors of **frog**: frogs, toad, litoria, leptodactylidae, rana, lizard, eleutherodactylus
- Presence of much rarer synonyms is an example of why practitioners may embed *near* the sphere, with slight incorporation of popularity into norm, and penalize niche recommendations.

Nearest neighbors from GloVe embeddings

`https://nlp.stanford.edu/projects/glove/`

- The typical training set is Wikipedia (historically, also non-free Gigaword 5).
- Main example from website is nearest neighbors of **frog**: frogs, toad, litoria, leptodactylidae, rana, lizard, eleutherodactylus
- Presence of much rarer synonyms is an example of why practitioners may embed *near* the sphere, with slight incorporation of popularity into norm, and penalize niche recommendations.

Nearest neighbors from GloVe embeddings

`https://nlp.stanford.edu/projects/glove/`

- The typical training set is Wikipedia (historically, also non-free Gigaword 5).
- Main example from website is nearest neighbors of **frog**: frogs, toad, litoria, leptodactylidae, rana, lizard, eleutherodactylus
- Presence of much rarer synonyms is an example of why practitioners may embed *near* the sphere, with slight incorporation of popularity into norm, and penalize niche recommendations.

Stages of a typical recommender

1. **Retrieval**: Return ~ 300 nearest neighbors using cosine similarity or its analogue.
2. **Reranking**: Fine-tune the ordering of the retrieved list using a classifier which approximately provides a $<$ operator for pairs.
3. **Diversification**: Return a list of, say, 10 results which balances relevance with diversity (e.g., via greedily sampling a Determinantal Point Process).

Today and tomorrow, we will be talking about solvers related to the **retrieval** and **diversification** phases. **Reranking** is now typically a DNN (though, so to is map from items to embeddings).

Stages of a typical recommender

1. **Retrieval**: Return ~ 300 nearest neighbors using cosine similarity or its analogue.
2. **Reranking**: Fine-tune the ordering of the retrieved list using a classifier which approximately provides a $<$ operator for pairs.
3. **Diversification**: Return a list of, say, 10 results which balances relevance with diversity (e.g., via greedily sampling a Determinantal Point Process).

Today and tomorrow, we will be talking about solvers related to the **retrieval** and **diversification** phases. **Reranking** is now typically a DNN (though, so to is map from items to embeddings).

Stages of a typical recommender

1. **Retrieval**: Return ~ 300 nearest neighbors using cosine similarity or its analogue.
2. **Reranking**: Fine-tune the ordering of the retrieved list using a classifier which approximately provides a $<$ operator for pairs.
3. **Diversification**: Return a list of, say, 10 results which balances relevance with diversity (e.g., via greedily sampling a Determinantal Point Process).

Today and tomorrow, we will be talking about solvers related to the **retrieval** and **diversification** phases. **Reranking** is now typically a DNN (though, so to is map from items to embeddings).

Stages of a typical recommender

1. **Retrieval**: Return ~ 300 nearest neighbors using cosine similarity or its analogue.
2. **Reranking**: Fine-tune the ordering of the retrieved list using a classifier which approximately provides a $<$ operator for pairs.
3. **Diversification**: Return a list of, say, 10 results which balances relevance with diversity (e.g., via greedily sampling a Determinantal Point Process).

Today and tomorrow, we will be talking about solvers related to the **retrieval** and **diversification** phases. **Reranking** is now typically a DNN (though, so to is map from items to embeddings).

Overview of lectures

Today:

- **Lecture**: Intro to Recommender Systems
- **Lecture**: Intro to Determinantal Point Processes
- **Lab**: Dense Determinantal Point Processes
- **Lecture**: Sparse-direct Factorization and DPPs

Tomorrow:

- **Lecture**: Solvers for Gaussian Low-Rank Inference
- **Lab**: Word embeddings via alternating weighted least squares
- **Lecture**: Conic automorphisms and equilibration of alternating WLS
- **Lab**: Implementing an equilibrated AWLS recommender with diversification

Overview of lectures

Today:

- **Lecture**: Intro to Recommender Systems
- **Lecture**: Intro to Determinantal Point Processes
- **Lab**: Dense Determinantal Point Processes
- **Lecture**: Sparse-direct Factorization and DPPs

Tomorrow:

- **Lecture**: Solvers for Gaussian Low-Rank Inference
- **Lab**: Word embeddings via alternating weighted least squares
- **Lecture**: Conic automorphisms and equilibration of alternating WLS
- **Lab**: Implementing an equilibrated AWLS recommender with diversification

Overview of lectures

Today:

- **Lecture**: Intro to Recommender Systems
- **Lecture**: Intro to Determinantal Point Processes
- **Lab**: Dense Determinantal Point Processes
- **Lecture**: Sparse-direct Factorization and DPPs

Tomorrow:

- **Lecture**: Solvers for Gaussian Low-Rank Inference
- **Lab**: Word embeddings via alternating weighted least squares
- **Lecture**: Conic automorphisms and equilibration of alternating WLS
- **Lab**: Implementing an equilibrated AWLS recommender with diversification

Overview of lectures

Today:

- **Lecture**: Intro to Recommender Systems
- **Lecture**: Intro to Determinantal Point Processes
- **Lab**: Dense Determinantal Point Processes
- **Lecture**: Sparse-direct Factorization and DPPs

Tomorrow:

- **Lecture**: Solvers for Gaussian Low-Rank Inference
- **Lab**: Word embeddings via alternating weighted least squares
- **Lecture**: Conic automorphisms and equilibration of alternating WLS
- **Lab**: Implementing an equilibrated AWLS recommender with diversification

Overview of lectures

Today:

- **Lecture**: Intro to Recommender Systems
- **Lecture**: Intro to Determinantal Point Processes
- **Lab**: Dense Determinantal Point Processes
- **Lecture**: Sparse-direct Factorization and DPPs

Tomorrow:

- **Lecture**: Solvers for Gaussian Low-Rank Inference
- **Lab**: Word embeddings via alternating weighted least squares
- **Lecture**: Conic automorphisms and equilibration of alternating WLS
- **Lab**: Implementing an equilibrated AWLS recommender with diversification

Overview of lectures

Today:

- **Lecture**: Intro to Recommender Systems
- **Lecture**: Intro to Determinantal Point Processes
- **Lab**: Dense Determinantal Point Processes
- **Lecture**: Sparse-direct Factorization and DPPs

Tomorrow:

- **Lecture**: Solvers for Gaussian Low-Rank Inference
- **Lab**: Word embeddings via alternating weighted least squares
- **Lecture**: Conic automorphisms and equilibration of alternating WLS
- **Lab**: Implementing an equilibrated AWLS recommender with diversification

Overview of lectures

Today:

- **Lecture**: Intro to Recommender Systems
- **Lecture**: Intro to Determinantal Point Processes
- **Lab**: Dense Determinantal Point Processes
- **Lecture**: Sparse-direct Factorization and DPPs

Tomorrow:

- **Lecture**: Solvers for Gaussian Low-Rank Inference
- **Lab**: Word embeddings via alternating weighted least squares
- **Lecture**: Conic automorphisms and equilibration of alternating WLS
- **Lab**: Implementing an equilibrated AWLS recommender with diversification

Overview of lectures

Today:

- **Lecture**: Intro to Recommender Systems
- **Lecture**: Intro to Determinantal Point Processes
- **Lab**: Dense Determinantal Point Processes
- **Lecture**: Sparse-direct Factorization and DPPs

Tomorrow:

- **Lecture**: Solvers for Gaussian Low-Rank Inference
- **Lab**: Word embeddings via alternating weighted least squares
- **Lecture**: Conic automorphisms and equilibration of alternating WLS
- **Lab**: Implementing an equilibrated AWLS recommender with diversification

Intro to Determinantal Point Processes

- Definition as distribution over subsets of a ground set.
- Aztec diamond domino tilings
- Star space and Uniform Spanning Tree processes
- (Sampling from) Determinantal Projection Processes
- Classical sampling algorithm for Hermitian DPPs
- Equivalence classes of DPPs and non-Hermitian DPPs
- Schur complements and conditional DPPs
- High-performance DPP sampling via modified matrix factorization

Intro to Determinantal Point Processes

- Definition as distribution over subsets of a ground set.
- Aztec diamond domino tilings
- Star space and Uniform Spanning Tree processes
- (Sampling from) Determinantal Projection Processes
- Classical sampling algorithm for Hermitian DPPs
- Equivalence classes of DPPs and non-Hermitian DPPs
- Schur complements and conditional DPPs
- High-performance DPP sampling via modified matrix factorization

Intro to Determinantal Point Processes

- Definition as distribution over subsets of a ground set.
- Aztec diamond domino tilings
- Star space and Uniform Spanning Tree processes
- (Sampling from) Determinantal Projection Processes
- Classical sampling algorithm for Hermitian DPPs
- Equivalence classes of DPPs and non-Hermitian DPPs
- Schur complements and conditional DPPs
- High-performance DPP sampling via modified matrix factorization

Intro to Determinantal Point Processes

- Definition as distribution over subsets of a ground set.
- Aztec diamond domino tilings
- Star space and Uniform Spanning Tree processes
- (Sampling from) Determinantal Projection Processes
- Classical sampling algorithm for Hermitian DPPs
- Equivalence classes of DPPs and non-Hermitian DPPs
- Schur complements and conditional DPPs
- High-performance DPP sampling via modified matrix factorization

Intro to Determinantal Point Processes

- Definition as distribution over subsets of a ground set.
- Aztec diamond domino tilings
- Star space and Uniform Spanning Tree processes
- (Sampling from) Determinantal Projection Processes
- Classical sampling algorithm for Hermitian DPPs
- Equivalence classes of DPPs and non-Hermitian DPPs
- Schur complements and conditional DPPs
- High-performance DPP sampling via modified matrix factorization

Intro to Determinantal Point Processes

- Definition as distribution over subsets of a ground set.
- Aztec diamond domino tilings
- Star space and Uniform Spanning Tree processes
- (Sampling from) Determinantal Projection Processes
- Classical sampling algorithm for Hermitian DPPs
- Equivalence classes of DPPs and non-Hermitian DPPs
- Schur complements and conditional DPPs
- High-performance DPP sampling via modified matrix factorization

Intro to Determinantal Point Processes

- Definition as distribution over subsets of a ground set.
- Aztec diamond domino tilings
- Star space and Uniform Spanning Tree processes
- (Sampling from) Determinantal Projection Processes
- Classical sampling algorithm for Hermitian DPPs
- Equivalence classes of DPPs and non-Hermitian DPPs
- Schur complements and conditional DPPs
- High-performance DPP sampling via modified matrix factorization

Intro to Determinantal Point Processes

- Definition as distribution over subsets of a ground set.
- Aztec diamond domino tilings
- Star space and Uniform Spanning Tree processes
- (Sampling from) Determinantal Projection Processes
- Classical sampling algorithm for Hermitian DPPs
- Equivalence classes of DPPs and non-Hermitian DPPs
- Schur complements and conditional DPPs
- High-performance DPP sampling via modified matrix factorization

Lab: Dense DPPs

We will implement (in Python):

- A Hermitian DPP sampler, and
- An elementary DPP sampler,

then apply them to uniformly sampling spanning trees from a box in $\mathbb{Z}^2$ (by constructing the Gramian of an orthonormal basis for the graph's star space).

Sparse-direct Factorization and DPPs

- Introduction to sparse-direct methods
- Approximate Minimum Degree reorderings
- Nested Dissection reorderings
- Formation of the elimination forest
- Formation of fundamental supernodes
- Supernode relaxation
- Numerical factorization/sampling
- Parallelization

Sparse-direct Factorization and DPPs

- Introduction to sparse-direct methods
- Approximate Minimum Degree reorderings
- Nested Dissection reorderings
- Formation of the elimination forest
- Formation of fundamental supernodes
- Supernode relaxation
- Numerical factorization/sampling
- Parallelization

Sparse-direct Factorization and DPPs

- Introduction to sparse-direct methods
- Approximate Minimum Degree reorderings
- Nested Dissection reorderings
- Formation of the elimination forest
- Formation of fundamental supernodes
- Supernode relaxation
- Numerical factorization/sampling
- Parallelization

Sparse-direct Factorization and DPPs

- Introduction to sparse-direct methods
- Approximate Minimum Degree reorderings
- Nested Dissection reorderings
- Formation of the elimination forest
- Formation of fundamental supernodes
- Supernode relaxation
- Numerical factorization/sampling
- Parallelization

Sparse-direct Factorization and DPPs

- Introduction to sparse-direct methods
- Approximate Minimum Degree reorderings
- Nested Dissection reorderings
- Formation of the elimination forest
- Formation of fundamental supernodes
- Supernode relaxation
- Numerical factorization/sampling
- Parallelization

Sparse-direct Factorization and DPPs

- Introduction to sparse-direct methods
- Approximate Minimum Degree reorderings
- Nested Dissection reorderings
- Formation of the elimination forest
- Formation of fundamental supernodes
- Supernode relaxation
- Numerical factorization/sampling
- Parallelization

Sparse-direct Factorization and DPPs

- Introduction to sparse-direct methods
- Approximate Minimum Degree reorderings
- Nested Dissection reorderings
- Formation of the elimination forest
- Formation of fundamental supernodes
- Supernode relaxation
- Numerical factorization/sampling
- Parallelization

Sparse-direct Factorization and DPPs

- Introduction to sparse-direct methods
- Approximate Minimum Degree reorderings
- Nested Dissection reorderings
- Formation of the elimination forest
- Formation of fundamental supernodes
- Supernode relaxation
- Numerical factorization/sampling
- Parallelization

Solvers for Gaussian Low-Rank Inference

- Bayesian interpretation of SVD objective function
- Bayesian interpretation of SVD on just the observed entries
- Bayesian interpretation of Gap SVD
- (Block) coordinate descent solver
- Fast alternating weighted least squares with constant background
- Column rescalings
- Generalizations of cosine similarity and relaxing off sphere
- Background on correspondence between symmetric cones and formally real Jordan algebras.

Solvers for Gaussian Low-Rank Inference

- Bayesian interpretation of SVD objective function
- Bayesian interpretation of SVD on just the observed entries
- Bayesian interpretation of Gap SVD
- (Block) coordinate descent solver
- Fast alternating weighted least squares with constant background
- Column rescalings
- Generalizations of cosine similarity and relaxing off sphere
- Background on correspondence between symmetric cones and formally real Jordan algebras.

Solvers for Gaussian Low-Rank Inference

- Bayesian interpretation of SVD objective function
- Bayesian interpretation of SVD on just the observed entries
- Bayesian interpretation of Gap SVD
- (Block) coordinate descent solver
- Fast alternating weighted least squares with constant background
- Column rescalings
- Generalizations of cosine similarity and relaxing off sphere
- Background on correspondence between symmetric cones and formally real Jordan algebras.

Solvers for Gaussian Low-Rank Inference

- Bayesian interpretation of SVD objective function
- Bayesian interpretation of SVD on just the observed entries
- Bayesian interpretation of Gap SVD
- (Block) coordinate descent solver
- Fast alternating weighted least squares with constant background
- Column rescalings
- Generalizations of cosine similarity and relaxing off sphere
- Background on correspondence between symmetric cones and formally real Jordan algebras.

Solvers for Gaussian Low-Rank Inference

- Bayesian interpretation of SVD objective function
- Bayesian interpretation of SVD on just the observed entries
- Bayesian interpretation of Gap SVD
- (Block) coordinate descent solver
- Fast alternating weighted least squares with constant background
- Column rescalings
- Generalizations of cosine similarity and relaxing off sphere
- Background on correspondence between symmetric cones and formally real Jordan algebras.

Solvers for Gaussian Low-Rank Inference

- Bayesian interpretation of SVD objective function
- Bayesian interpretation of SVD on just the observed entries
- Bayesian interpretation of Gap SVD
- (Block) coordinate descent solver
- Fast alternating weighted least squares with constant background
- Column rescalings
- Generalizations of cosine similarity and relaxing off sphere
- Background on correspondence between symmetric cones and formally real Jordan algebras.

Solvers for Gaussian Low-Rank Inference

- Bayesian interpretation of SVD objective function
- Bayesian interpretation of SVD on just the observed entries
- Bayesian interpretation of Gap SVD
- (Block) coordinate descent solver
- Fast alternating weighted least squares with constant background
- Column rescalings
- Generalizations of cosine similarity and relaxing off sphere
- Background on correspondence between symmetric cones and formally real Jordan algebras.

Solvers for Gaussian Low-Rank Inference

- Bayesian interpretation of SVD objective function
- Bayesian interpretation of SVD on just the observed entries
- Bayesian interpretation of Gap SVD
- (Block) coordinate descent solver
- Fast alternating weighted least squares with constant background
- Column rescalings
- Generalizations of cosine similarity and relaxing off sphere
- Background on correspondence between symmetric cones and formally real Jordan algebras.

Lab: Word embeddings via alternating WLS

We will construct an alternating weighted least squares solver in python to train word embeddings from a subset of Wikipedia.

`https://github.com/attardi/wikiextractor` takes several hours but generates 13 GB of plain text from Wikipedia.

Hotelwork: Either run this script overnight on `https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2` or download a bzip2 of 10 percent of the results posted at: `https://send.firefox.com/download/a33e163f35fcb5d4/#0T9goiJliiWlNKlkHVHL_Q` or `https://bit.ly/31ACQ8i`.

Further, write a python function for traversing this hierarchy of plain text files with a given window size to produce a dictionary of cooccurrence scores (keyed on the source terms with the value being the array of target/cooccurrence pairs).

Then write a separate routine to return a filtered dictionary which drops coocurrences with sufficiently small scores and keeps only the dominant n words. (Bonus: Also incorporate bigrams.)

Lab: Word embeddings via alternating WLS

We will construct an alternating weighted least squares solver in python to train word embeddings from a subset of Wikipedia.

`https://github.com/attardi/wikiextractor` takes several hours but generates 13 GB of plain text from Wikipedia.

Hotelwork: Either run this script overnight on `https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2` or download a bzip2 of 10 percent of the results posted at: `https://send.firefox.com/download/a33e163f35fcb5d4/#0T9goiJliiWlNKlkHVHL_Q` or `https://bit.ly/31ACQ8i`.

Further, write a python function for traversing this hierarchy of plain text files with a given window size to produce a dictionary of cooccurrence scores (keyed on the source terms with the value being the array of target/cooccurrence pairs).

Then write a separate routine to return a filtered dictionary which drops coocurrences with sufficiently small scores and keeps only the dominant n words. (Bonus: Also incorporate bigrams.)

Lab: Word embeddings via alternating WLS

We will construct an alternating weighted least squares solver in python to train word embeddings from a subset of Wikipedia.

`https://github.com/attardi/wikiextractor` takes several hours but generates 13 GB of plain text from Wikipedia.

Hotelwork: Either run this script overnight on `https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2` or download a bzip2 of 10 percent of the results posted at: `https://send.firefox.com/download/a33e163f35fcb5d4/#0T9goiJliiWlNKlkHVHL_Q` or `https://bit.ly/31ACQ8i`.

Further, write a python function for traversing this hierarchy of plain text files with a given window size to produce a dictionary of cooccurrence scores (keyed on the source terms with the value being the array of target/cooccurrence pairs).

Then write a separate routine to return a filtered dictionary which drops coocurrences with sufficiently small scores and keeps only the dominant n words. (Bonus: Also incorporate bigrams.)

Lab: Word embeddings via alternating WLS

We will construct an alternating weighted least squares solver in python to train word embeddings from a subset of Wikipedia.

`https://github.com/attardi/wikiextractor` takes several hours but generates 13 GB of plain text from Wikipedia.

Hotelwork: Either run this script overnight on `https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2` or download a bzip2 of 10 percent of the results posted at: `https://send.firefox.com/download/a33e163f35fcb5d4/#0T9goiJliiWlNKlkHVHL_Q` or `https://bit.ly/31ACQ8i`.

Further, write a python function for traversing this hierarchy of plain text files with a given window size to produce a dictionary of cooccurrence scores (keyed on the source terms with the value being the array of target/cooccurrence pairs).

Then write a separate routine to return a filtered dictionary which drops coocurrences with sufficiently small scores and keeps only the dominant n words. (Bonus: Also incorporate bigrams.)

Conic automorphisms and equilibration of alternating WLS

- Why Gramians of optimal solutions are equal
- Equilibrating the factors via conic automorphisms
- Relationship to conic Interior Point Methods
- Handling ill-conditioned/singular Gramians
- A Jordan-algebraic interpretation

Conic automorphisms and equilibration of alternating WLS

- Why Gramians of optimal solutions are equal
- Equilibrating the factors via conic automorphisms
- Relationship to conic Interior Point Methods
- Handling ill-conditioned/singular Gramians
- A Jordan-algebraic interpretation

Conic automorphisms and equilibration of alternating WLS

- Why Gramians of optimal solutions are equal
- Equilibrating the factors via conic automorphisms
- Relationship to conic Interior Point Methods
- Handling ill-conditioned/singular Gramians
- A Jordan-algebraic interpretation

Conic automorphisms and equilibration of alternating WLS

- Why Gramians of optimal solutions are equal
- Equilibrating the factors via conic automorphisms
- Relationship to conic Interior Point Methods
- Handling ill-conditioned/singular Gramians
- A Jordan-algebraic interpretation

Conic automorphisms and equilibration of alternating WLS

- Why Gramians of optimal solutions are equal
- Equilibrating the factors via conic automorphisms
- Relationship to conic Interior Point Methods
- Handling ill-conditioned/singular Gramians
- A Jordan-algebraic interpretation

Lab: Synonyms via equilibrated AWLS and diversification

We will incorporate a Determinantal Point Process diversification process and Nesterov Todd equilibration into our AWLS method for word embeddings.

Some readings

Determinantal Point Processes:

- Kulesza and Taskar, Determinantal point processes for machine learning, arXiv:1207.6083.
- Hough, Krishnapur, Peres, and Virag, Determinantal Processes and Independence, arXiv:math/0503110.
- Poulson, High-performance sampling of generic Determinantal Point Processes, arxiv:1905.00165.

Background weights for recommenders:

- Pan and Scholz, Mind the gaps: weighting the unknowns in large-scale one-class collaborative filtering, KDD, 2009.

Jordan algebras and conic IPMs:

- Alizadeh and Goldfarb, Second-Order Cone Programming, 2001.
  `http://rutcor.rutgers.edu/~alizadeh/CLASSES/03sprNLP/Papers/allSurvey.pdf`

Discussion

Slides are available at:
hodgestar.com/G2S3/

Chatroom at:
`https://gitter.im/hodge_star/G2S3`