FEAT2VEC: SEMANTIC MODULARITY FOR CONTINUOUS REPRESENTATIONS OF DATA WITH FEATURES

Anonymous authors
Paper under double-blind review

ABSTRACT

Methods that calculate dense embeddings for features in unstructured data—such as words in a document—have proven to be very successful for knowledge representation. Surprisingly, very little work has focused on methods for structured datasets where there is more than one feature type—this is, datasets that have arbitrary features beyond words.

We study how to estimate continuous representations for multiple feature types within a dataset, where each feature type exists in a different higher-dimensional space. Feat2Vec is a novel method that calculates embeddings for data with multiple feature types enforcing *semantic modularity* across features—all different feature types exist in a common space. We demonstrate our work on two datasets. The first one is collected from a leading educational technology firm, and thus we are able to discover a common continuous space for entities such as universities, courses, textbooks and students. The second dataset is the public IMDB dataset, and we discover embeddings for entities that include actors, movies and producers. Our experiments suggest that Feat2Vec significantly outperforms existing algorithms that do not leverage the structure of the data.

1 PRELIMINARIES

Informally, in machine learning an *embedding* of a vector $\vec{x} \in \mathbb{R}^n$ is another vector $\vec{y} \in \mathbb{R}^r$ that has much lower dimensionality ($r \ll n$) than the original representation, and can be used to replace the original vector in downstream prediction tasks. Embeddings have multiple advantages, as they enable more efficient training (Mikolov et al., 2013b), and unsupervised learning (Schnabel et al., 2015). For example, when applied to text, semantically similar words are mapped to nearby points.

Word2Vec (Mikolov et al., 2013a;b) is an extremely successful software package that contains two embedding functions with the same domain and codomain:

$$\text{Word2Vec} : \left\{ \vec{x} \in \mathbb{R}^n \mapsto \vec{y} \in \mathbb{R}^r \right\} \tag{1}$$

Word2Vec is suited for calculating embeddings for datasets that consist of documents of words with a vocabulary size of n. Here, $\vec{x}$ is sparse because words (and categorical variables, in general) are modeled using one-hot encoding. In this paper we study how to generalize embedding functions for arbitrary datasets with multiple feature types by leveraging their internal structure. We call this approach Feat2Vec:

$$\text{Feat2Vec} : \left\{ \vec{x} \in \vec{\mathcal{F}} \mapsto \vec{y} \in \mathbb{R}^r \right\} \tag{2}$$

$$\vec{\mathcal{F}} = \left[\mathbb{R}^{d_1}, \mathbb{R}^{d_2}, \ldots, \mathbb{R}^{d_n} \right] \tag{3}$$

Here, $\vec{\mathcal{F}}$ is a structured feature set—where each dimension represents a different feature type. Notice that is possible to use Word2Vec on structured data by simply flattening the input. However, Feat2Vec leverage the features' structure to enforce *semantic modularity* across features. That is, the embedding of an observation is additive in respect to the embeddings of its feature types; thus, each feature type $\mathcal{F}_i$ must be projected into the same embedding space. Semantic modularity is useful because it enables making principled comparisons between different feature types. For example, on an educational dataset that contains many feature types, including students and courses, semantic modularity enables inferring course preferences for students.

2 Feat2Vec

The rest of this section describes the three parts of a Feat2Vec implementation. § 2.1 describes a model that infers an embedding from an observation. To learn this model we need positive and negative examples: a positive example is "semantic" and it is observed during training, but negative examples are not. This is similar how a positive example in Word2Vec are grammatical co-ocurrences, and negative examples are generated. § 2.2 describes our novel sampling strategy. Finally, in § 2.3 we describe how to learn a Feat2Vec model from data.

2.1 Structured Deep-In Factorization Machine

Levy & Goldberg (2014) showed that a Word2Vec model can be formalized as a Shallow Factorization Machine (Rendle, 2010) with two features types: a word and its context. This factorization model is a binary classifier that scores the likelihood of an observation $\vec{x} \in \mathbb{R}^n$ being labeled $y \in \{0, 1\}$, as proportional to the sum of the factorized pairwise interactions:

$$p\big(Y = 1|\vec{x}\big) = \sigma\left(\sum_{i=1}^{n}\sum_{j=i}^{n} \langle\, \vec{\beta}_i x_i,\, \vec{\beta}_j x_j \,\rangle\right) \tag{4}$$

Here, Y is the random variable that defines whether the observation is "semantic" (which in practice means whether it occurs in the training data), $\vec{\beta}_i$ is a rank-r vector of factors, and σ is a sigmoid:

$$\sigma(x) = \log\left(\frac{\exp(x)}{\exp(x+1)}\right) \tag{5}$$

However, features may have some structure that is known beforehand. For example, consider multiple discrete entity types, like universities, students, and books. To model this data, a Shallow Factorization Machine would ignore the structure and simply concatenate the one-hot encoding representation of each entity into a single feature vector. Feat2Vec relies on an novel extension to factorization called Structured Deep-In Factorization Machine, which we describe in detail in a companion paper (Anonymized, Under review). While Shallow Factorization Machine learns an embedding per feature, the Structured Deep-In model allows greater flexibility. For example, consider using images or text on these factorization models. The shallow model learns an embedding for each word, or an embedding per pixel. The structured model enables higher-level of abstraction and flexibility, and it can learn an embedding per passage of text, or an embedding per image.

Structured Deep-In Factorization Machine inputs $\vec{\kappa}$ as a vector of vectors that define the structure of the groups of features. Each entry i is a vector of size d_i and it is used to represent the feature types of $\mathcal{F}_i$. In a shallow model, a feature interacts with all other features, but in the structured model they only interact with features from different groups. For example, consider a model with four features. If we define $\vec{\kappa} = \big[\,[1,2],[3,4]\,\big]$, the structure of the model would not allow x_1 to interact with x_2, or x_3 to interact with x_4. Additionally, for each group of features, the model allows applying a $d_i \times r$ feature extraction function ϕ_i. More formally, this is:

$$p\big(Y = 1|\vec{x}; \vec{\phi}, \vec{\kappa}\big) = \sigma\Big(s(\vec{x}, \vec{\phi})\Big) = \sigma\left(\sum_{i=1}^{|\vec{\kappa}|}\sum_{j=i}^{|\vec{\kappa}|}\big\langle\, \phi_i(\vec{x}_{\vec{\kappa}_i}),\ \phi_j(\vec{x}_{\vec{\kappa}_j})\,\big\rangle\right) \tag{6}$$

In this notation, $\vec{x}_{\vec{\kappa}_i}$ is a subvector that contains all of the features that belong to the group κ_i, and $s(\vec{x}, \vec{\phi})$ is the score function. Thus, $\vec{x}_{\vec{\kappa}_i} = \big[\, x_{i_a} | a \in \{1, 2, \ldots, |\kappa_i|\}\,\big]$. The simplest implementation for ϕ_i is a linear fully-connected layer, where the output of the r-th entry is:

$$\phi_i\big(\vec{x}_i; \vec{\beta}\big)_r = \sum_{a=1}^{d_i} \beta_{r_a} x_{i_a}$$

We build the Feat2Vec embedding for an observation $\vec{x}$ as:

$$\text{Feat2Vec}(x) = \big[\ \phi_q(\vec{x}_{\vec{\kappa}_1}),\ \phi_1(\vec{x}_{\vec{\kappa}_2}),\ \ldots,\ \phi_n(\vec{x}_{\vec{\kappa}_n})\ \big] \tag{7}$$

2.2 SAMPLING

The training dataset for a Feat2Vec model consists of only semantic observations. In natural language, these would be documents written by humans. Since Structured Deep-In Factorization Machine (Equation 6) requires positive and negative examples, we also need to supply observations that are not semantic. Consider a feature type $\mathcal{F}_i \in \mathbb{R}^{d_i}$, that exists in very high dimensional space (d_i is large). For example, this could happen because we are modeling with one-hot encoding a categorical variable with large number of possible values. In such scenario, it is overwhelmingly costly to feed the model all negative labels, particularly if the model is fairly sparse.

A shortcut around this is a concept known as *implicit sampling*, where instead of using all of the possible negative labels, one simply samples a fixed number (k) from the set of possible negative labels for each positively labelled record. Word2Vec makes use of an algorithm called Negative Sampling (Dyer, 2014). In short, their approach samples a negative observation w from a noise distribution q:

$$\begin{aligned} \text{Draw } w &\sim p(W|Y=0,X) && (8)\\ &= q(\{w_1, w_2, \dots, w_n\}, \alpha) && (9) \end{aligned}$$

Here q is the empirical frequency of w in the dataset, exponentiated by α, a flattening hyper-parameter:

$$q(W,\alpha) = \frac{c_X(w)^\alpha}{\sum\limits_{w' \in W} c_X(w')^\alpha} \tag{10}$$

Here, $c_X(w)$ is the number of times a feature w appeared in the training dataset X. With $\alpha = 1$, the noise distribution corresponds to the empirical frequency distribution, and with $\alpha = 0$ to the uniform distribution. Intermediate values are intended to be continuous progressions to these two extremes.

Our main contribution is introducing a new implicit sampling method that enables learning embedding for structured feature sets. We can learn the correlation of features within a dataset by imputing negative labels, simply by generating "unobserved" records as our negative sample. Unlike Word2Vec, we do not constraint features types to be words. Features types can be individual numeric columns, but they need not to be. By defining feature types as groups of subfeatures (using κ parameter in Equation 6), the model can reason on more complex entities. For example, in our experiments on a movie dataset, we use a "genre" feature type, where we group non-mutually exclusive indicators for comedy, action, and drama films under one umbrella. For more involved feature types (e.g., an image), one would need to define a function ϕ that builds intermediate layers to map the entity to an embedding .

We start with a dataset S^+ of records with p feature types. We then mark all observed records in the training set as positive examples. For each positive record, we generate k negative labels using the following 2-step algorithm:

Algorithm 1 Implicit sampling algorithm for Feat2Vec

```
1:  function FEAT2VEC_SAMPLE($S^+, k, \alpha_1, \alpha_2$)
2:      $S^- \leftarrow \emptyset$
3:      for $s \in S^+$ do
4:          Draw a random feature type $\kappa_i \sim q_1(\{d_i\}_{i=1}^{p}, \alpha_1)$
5:          for $j \in \{1, \dots, k\}$ do
6:              $\vec{e} \leftarrow \vec{x}^{(s)}$                ▷ set initially to be equal to a positive sample
7:              Draw a random (sub)feature $\vec{r} \sim q_2(\mathrm{X}_{\kappa_i}, \alpha_2)$
8:              $\vec{e}_{\kappa_i} \leftarrow \vec{r}$                ▷ substitute the $i$-th feature type with a sampled one
9:              $S^- \leftarrow S^- + \{\vec{e}\}$
10:         end for
11:     end for
12:     return $S^-$
13: end function
```

Here $q_2(\mathrm{X}_{\kappa_i}, \alpha_2)$ is the "flattened" empirical multionomial distribution over the frequency of the feature values of feature κ_i, as in the negative sampling procedure used in Word2Vec :

$$x_j \sim q_2(\mathrm{X}_{\kappa_i}, \alpha_2), \text{ where } P_{q_2}(x_j = x|\mathrm{X}_{\kappa_i}, \alpha_2) \;=\; \frac{c_X(x)^{\alpha_2}}{\sum_{x' \in X_{\kappa_i}} c_X(x')^{\alpha_2}}, \alpha \in [0, 1]$$

Where $P_q(x)$ denotes the probability of x under a distribution q. Analogously , $q_1(\{d_i\}_{i=1}^p, \alpha_1)$ draws a feature to negatively sample from a multinomial distribution with probabilitites proportional a feature's complexity, d_i. Here, by complexity, we mean the number of parameters the algorithm learns associated with a feature group:

$$P_{q_1}(\kappa_i|\{d_i\}_{i=1}^p, \alpha_1) \;=\; \frac{d_i^{\alpha_1}}{\sum_{k=1}^p d_k^{\alpha_1}}$$

d_i may be the dimensionality of a feature, but might also capture other parameters used for learning embeddings, where applicable. For example, if we have multiple intermediate layers to process a set of pixels embeddings into an "image embedding" (such as deep convolutional layers), these parameters would count towards d_i.

Explained in words, our negative sampling method is to randomly select one of the features κ from the noise distribution $q_1(.)$, and, conditional on the feature to be resampled, choose another value x_j for feature κ from a noise distribution $q_2(.)$. In our application, we use the same class of noise distributions (flattened multinomial) for both levels of sampling, but this need not necessarily be the case.

This method will sometimes by chance generate negatively labeled samples that *do* exist in our sample of observed records. For this reason, among others, we employ a technique known as noise contrastive estimation (Gutmann & Hyvärinen, 2010) which uses basic probability laws to adjust the structural statistical model $p(Y = 1|\vec{\phi}, \vec{x})$ to account for the possibility of random negative labels that appear identical to positively labeled data. An additional burden of this method, however, is we need to learn additional nuisance parameters $Z_{\vec{x}}$ for each unique record type $\vec{x}$ that transform the score function $s(.)$ into a well-behaved probability distribution that integrates to 1. This introduces an astronomical amount of new parameters and greatly increase the complexity of the model. Instead of estimating these, we appeal to the work of (Mnih & Teh, 2012), who show in the context of language models that setting the $Z_{\vec{x}} = 1$ in advance effectively does not change the performance of the model. [1] Written explicitly, the new structural probability model is:

$$\tilde{p}(Y = 1|\vec{\phi}, \vec{x_i}) = \frac{\exp(s(\vec{x}_i, \vec{\phi}))}{\exp(s(\vec{x}_i, \vec{\phi})) + P_q(\vec{x}_i|\alpha_1, \alpha_2)}$$

where $P_q(.)$ denotes the unconditional probability of a a record $\vec{x_i}$ being drawn from our negative sampling algorithm, and $s(x, \vec{\phi})$ is the score function for Factorization Machines, in the syntax of (Gutmann & Hyvärinen, 2010). We can show, with relatively little effort,that our 2-step sampler has some interesting theoretical properties. Namely, the embeddings learned under this model will be equivalent to a convex combination of the embeddings learned from p individual Factorization Machines.

2.3 Learning From data

We optimize Noise Contrastive Estimation(Gutmann & Hyvärinen, 2010) loss, which uses basic probability laws to adjust the structural statistical model $p(Y = 1|\vec{\phi}, \vec{x})$ to account for the possibility of randomly generated negative labels that appear identical to positively labeled data. Written explicitly, the new structural probability model is just:

$$\tilde{p}(Y = 1|\vec{\phi}, \vec{x_i}) = \frac{\exp(s(\vec{x}_i, \vec{\phi}))}{\exp(s(\vec{x}_i, \vec{\phi})) + P_q(\vec{x}_i|\alpha_1, \alpha_2)} \times Z_{\vec{x}} \tag{11}$$

[1](Mnih & Teh, 2012) suggests one can set the normalizing constant to 1 because the neural net model has enough free parameters that it will effectively just learn the probabilities itself so that it does not over/under predict the probabilities on average (since that will result in penalties on the loss function). Note that while we follow the same procedure, one could set $Z_{\vec{x}}$ to any positive value in advance and the same logic would follow, if one was worried about astronomically low probabilities.

where $P_q(.)$ denotes the unconditional probability of a a record $\vec{x_i}$ being drawn from our implicit sampling algorithm, $s(x, \vec{\phi})$ is the score function for Factorization Machines, in the syntax of (Gutmann & Hyvärinen, 2010), and $Z_{\vec{x}_i}$ is a positive learned parameter to guarantee that the probability distribution integrates to 1:

$$\sum_{\vec{x}_i \in \mathrm{X}} Z_{\vec{x}_i} \times \frac{\exp(s(\vec{x}_i, \vec{\phi}))}{\exp(s(\vec{x}_i, \vec{\phi})) + P_q(\vec{x}_i|\alpha_1, \alpha_2)} = 1$$

This additional parameter for each unique record,$Z_{\vec{x}_i}$, introduces an astronomical amount of new parameters and greatly increase the complexity of the model. Instead of estimating these, we appeal to the work of (Mnih & Teh, 2012), who show in the context of language models that setting the $Z_{\vec{x}} = 1$ in advance effectively does not change the performance of the model.[2]

We can show, with relatively little effort,that our 2-step sampler has some interesting theoretical properties. Namely, the embeddings learned under this model will be equivalent to a convex combination of the embeddings learned from p individual Factorization Machines that also use an NCE loss:

Theorem 1. *The embeddings learned with* Feat2Vec *are a convex combination of the embeddings learned from* p *targeted Factorization Machines for each feature in the data.*

See the proof in the Appendix.

3 Limitations

Continuous features require a sensible feature function, and future work could focus on them.

Feature engineering (grouping) is important.

4 Empirical Results

In order to test the relative performance of our learned embeddings, we train our Feat2Vec algorithm on an unstructured dataset, and compare its performance in a targeted ranking task to Word2Vec. We use a publicly available dataset. Ex ante, it is unclear how to evaluate the performance of an "unsupervised" embedding algorithm, but we felt that a reasonable task would be to assess the similarity of trained embeddings using unseen records in a left-out dataset. We discuss the evaluation procedure in further detail below.

We use the publicly accessible IMDB database on movies[3]. It contains information on writers, directors, and principal cast members (actors) attached to each film, along with film metadata such as length, IMDB rating, and movie title. Each record corresponds to a single film. While the IMDB database contains information on TV series and other media types, we focus only on data on films. Table 1 displays the feature types we use from this dataset.

We now discuss our modeling choices for this particular implementation of Feat2Vec. For all of the category variables, we learn a unique d-dimensional embedding for each entity. When there are multiple categories per record, we sum the individual entity embeddings in a "bag-of-words" style representation to a single vector representing that feature's information content. For example, when there are multiple cast members, we aggregate the embeddings for each actor to a single "actor" embedding, that is then interacted with embeddings extracted from all other feature groups. To feed thsee sequence-type features into our Feat2Vec algorithm, we require a fixed length of categories for each feature type, and so truncate each to have no more than 10 categories per feature (and sometimes less if reasonable). This results in retaining the full set of information for well over

[2](Mnih & Teh, 2012) suggests one can set the normalizing constant to 1 because the neural net model has enough free parameters that it will effectively just learn the probabilities itself so that it does not over/under predict the probabilities on average (since that will result in penalties on the loss function). Note that while we follow the same procedure, one could set $Z_{\vec{x}}$ to any positive value in advance and the same logic would follow, if one was worried about astronomically low probabilities.

[3]More information can be found here: http://www.imdb.com/interfaces/

Table 1: Features from IMDB database used in Feat2Vec Algorithm

Feature	Feature Type	# of Categories	Example
Movie Title	Text Sequence	165,471	[”Ocean’s”,”Eleven”]
Movie Release Year	Category	217	2003
Is Adult Film?	Boolean	2	False
Directors	Category Sequence	174,382	[Martin Scorsese]
Writers	Category Sequence	244,241	[Stephen King, Frank Darabont]
Principal Cast Members	Category Sequence	1,104,280	[Samuel L Jackson, Uma Therman, John Travolta, . . .]
Genres	Category Sequence	28	[Action, Comedy]
Runtime (Minutes)	Real-Valued	-	90
IMDB Rating (0-10)	Real-Valued	-	5.5
# of IMDB Rating Votes	Real-Valued	-	150
Total # of Films		465,136	

95% of our observations. We pad the sequences with a “null” category whenever necessary to maintain a fixed length. For all real-valued features, we pass these features through a 3-layer feed-forward fully connected neural network that outputs a vector of dimension d, which we treat as the feature’s embedding. Each intermediate layer has d units with `relu` activation functions. These real-valued features highlight one of the advantages of the Feat2Vec algorithm: using a numeric value as an input, Feat2Vec can learn a highly nonlinear relation mapping a real number to our high-dimensional embedding space. In contrast, Word2Vec would ex ante be unable to know ex-ante that an IMDB rating of 5.5 is anything like an IMDB rating of 5.6; it would tokenize both values and treat them as independent words in a document. For the single text feature we use (the Movie Title), we remove preprocess the text by removing non alpha-numeric characters, pruning stopwords, and stemming the remaining words in the title text. We then follow a simple bag-of-words approach that learns an embedding for each unique word in our universe of movie titles, and sums these up, as we did for our categorical sequence variables. We note, however, that it would be trivial in our framework to process the text further by passing these word embeddings through sequential convolutional layers, to further extract high-level features of movie titles that may be relevant for learning embeddings.

For training Word2Vec embeddings on the IMDB database, we simply treat each movie record as a document, and tokenize all feature values by prepending each feature value in an observation with the name of the variable. So if a movie has a release year of 1986, we would input the word `releaseYear_1986` into that movie’s corresponding training document. We use the `cbow` Word2Vec algorithm and set the context window to encompass all other tokens in a document during training, since the text in this application is unordered. Whenever possible, we set identical hyperparameters for Word2Vec that we use in Feat2Vec.

For our evaluation, we leave out a random 10% sample of the IMDB dataset that contains a director[4] that appears at least twice in the database. We do this to guarantee that the set of directors in the left-out dataset appear during training at least once, so that each respective algorithm can learn something about the characteristics of these directors.

For training Feat2Vec, we set $\alpha_1 = 1/4$ and $\alpha_2 = 3/4$. We perform cross-validation on the loss function to determine the number of epochs to train, and then train the full training dataset with this number of epochs. While regularization of the embeddings during training is possible, this did not dramatically change results, so we ignore this dimension of hyperparameters. The Feat2Vec model is built on the python library Keras After training, we take our left out set of movies, and using the cast members associated with a film, attempt to predict the actual director the film was directed by. We take the sum of the cast member embeddings, and rank the directors by cosine similarity of their embeddings to the summed cast member vector. If there is a cast member in the test dataset who did not appear in the training data, we exclude them from the summation. The idea behind this evaluation is that if the embeddings learned in the training dataset are successful at extracting the underlying information about these entities from their movie projects, they should be able to predict future colloborations. We evaluate the rankings according to three metrics. The first is mean

[4]over 90% of the movies in the database have exactly one director, but in cases where there are multiple directors to a film, we use the first director listed in the IMDB dataset.

percentile rank (MPR), the average rank (standardized to be on a 0-100 percentile scale) of the actual directors in the test dataset:

$$MPR = \frac{1}{N}\sum_{i=1}^{N}\frac{R_i}{\max R}$$

where R_i is the rank of the director under our evaluation procedure for movie i. This measures on average how well we rank actual director-cast collaborations. A score of 0 would indicate perfect performance (i.e. top rank every test sample given), so a lower value is better under this metric. The next is mean reciprocal rank (MRR), which is similar to mean percentile rank, but weights rankings that are closer to rank 0 (the best rank possible) more:

$$MRR = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{R_i + 1}$$

The idea behind this metric is that, under MPR, an improvement in rankings under an algorithm from 10 to 1 will be equally weighted as an improvement from 49,103 to 49,92 in terms of calculating the MPR statistic. Clearly, the former improvement is more useful in a real-world context of submitting reccomendations to users, and so MRR attempts to correct for this by weighting rankings closer to 1 much higher. A higher value, unlike MPR, indicates better performance. The final metric we consider is top 1 precision, or the percent of test examples we successfully rank the actual director as 1 out of the 174,382 directors possible.

Table 2 presents the results from our evaluation. As is evident, Feat2Vec sizably outperforms Word2Vec along all metrics considered. Figure 1 shows the full distribution of rankings, rather than summary statistics, in the form of a CDF of all rankings calculated in the test dataset. The graphic makes it apparent for the vast majority of the ranking space, the rank CDF of Feat2Vec is to the left of Word2Vec, indicating a greater probability of a lower ranking under Feat2Vec. This is not, however, the case at the upper tail of ranking space, where it appears Word2Vec is superior. One might argue that while, overall, Feat2Vec appears to perform better, it is exactly that upper tail that actually matters most in practice, and so it would seem Word2Vec is more useful. We would argue that actually, intermediate rankings are still strong signals that our Feat2Vec algorithm is doing a better job of extracting information into embeddings, particularly those entities that appear sparsely in the training data and so are especially difficuly. But additionally, if we zoom-in on the absolute upper region of rankings (1 to 25), which might be a sensible length of ranks one might give an actual reccomendee, it is the case that up until rank 8 or so, Feat2Vec outperforms Word2Vec still, and so even on this metric, is preferred if we only care about top rankings.

Table 2: Algorithm Performances on Evaluation Ranking Task

Statistic	Feat2Vec	Word2Vec
Mean Percentile Rank	19.36%	24.15%
Mean Reciprocal Rank	0.0362	0.0297
Precision (Top 1)	2.43%	1.26%

5 RELATION TO PRIOR WORK

6 CONCLUSION

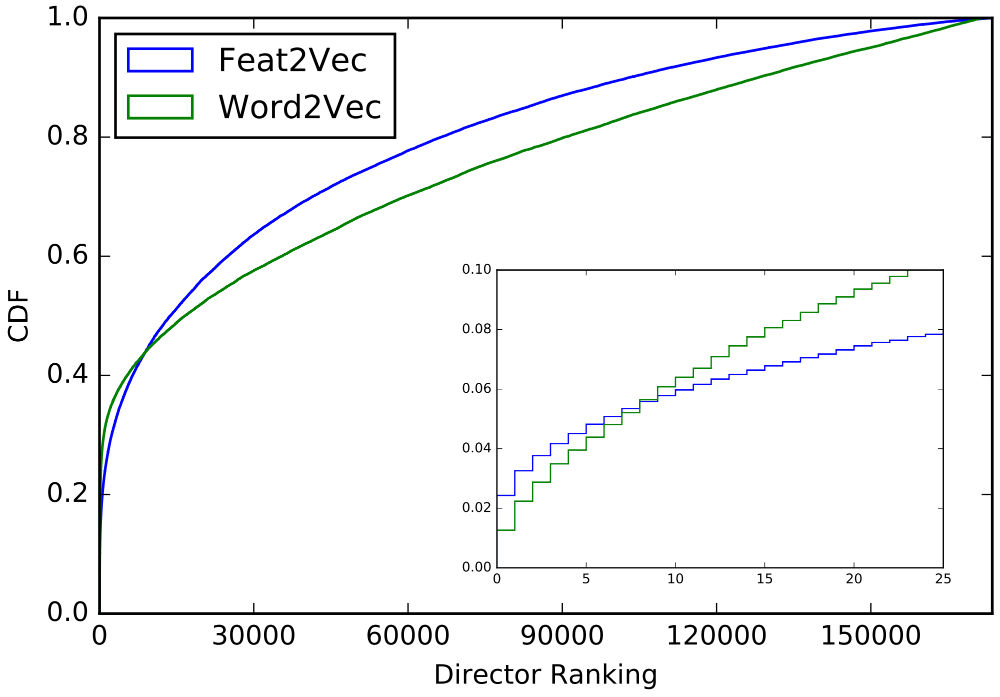

Figure 1: CDF of Test Data Director Rankings
(With Zoom-in to Top 25 Ranks)

A Appendixes

A.1 Proof to Theorem 1

Theorem 1. *The embeddings learned with* Feat2Vec *are a convex combination of the embeddings learned from* p *targeted Factorization Machines for each feature in the data.*

Proof. Let S_f^+ denote the positively labeled records whose corresponding negative samples resample feature f. We can express the loss function $L(.)$, the binary cross-entropy of the data given the Feat2Vec model, as follows:

$$
\begin{aligned}
L(\vec{\mathbf{x}}|\vec{\phi}) =& \frac{1}{|S^+|} \sum_{i \in S^+} \Big(\log(\tilde{p}(Y=1|\vec{\phi}, \vec{\mathbf{x}}_i)) + \sum_{\vec{\mathbf{x}}_j \sim q(.|\vec{\mathbf{x}}_i)}^{k} \log(\tilde{p}(Y=0|\vec{\phi}, \vec{\mathbf{x}}_j)) \Big) \\
=& \frac{1}{|S^+|} \sum_{i \in S^+} \Big(\log(\tilde{p}(Y=1|\vec{\phi}, \vec{\mathbf{x}}_i, i \in S_f) p(i \in S_f^+)) + \sum_{\vec{\mathbf{x}}_j \sim q(.|\vec{\mathbf{x}}_i)}^{k} \log(\tilde{p}(Y=0|\vec{\phi}, \vec{\mathbf{x}}_j, i \in S_f) p(i \in S_f^+)) \Big) \\
=& \frac{1}{|S^+|} \sum_{f=1}^{p} \sum_{i \in S_f^+} \Big(\log(\frac{e^{s(\vec{\mathbf{x}}_i, \vec{\phi})} p(i \in S_f^+)}{e^{s(\vec{\mathbf{x}}_i, \vec{\phi})} + P_q(\vec{\mathbf{x}}_i|\vec{\mathbf{x}}_i, i \in S_f^+)}) + \sum_{\vec{\mathbf{x}}_j \sim q(.|\vec{\mathbf{x}}_i, i \in S_f^+)}^{k} \log(\frac{P_q(\vec{\mathbf{x}}_j|\vec{\mathbf{x}}_i, i \in S_f) p(i \in S_f)}{e^{s(\vec{\mathbf{x}}_j, \vec{\phi})} + P_q(\vec{\mathbf{x}}_j|\vec{\mathbf{x}}_i, i \in S_f^+)}) \Big)
\end{aligned}
$$

Note now that $P_q(\vec{\mathbf{x}}_k|\vec{\mathbf{x}}_i, i \in S_f^+)$ is simply the probability of the record's feature value $\vec{\mathbf{x}}_{k,f}$ under the second step noise distribution $q_2(\mathbf{X_f}, \alpha_2)$: $P_q(\vec{\mathbf{x}}_k|\vec{\mathbf{x}}_i, i \in S_f^+) = P_{q_2}(\vec{\mathbf{x}}_{k,f})$, where k refers to record i or a record j negatively sampled from i.

$$=\frac{1}{|S^+|}\sum_{f=1}^{p}\sum_{i\in S_f^+}\Big(\log(\frac{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}p(i\in S_f^+)}{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}+P_{q_2}(\vec{\mathbf{x}}_{i,f})})+\sum_{\vec{\mathbf{x}}_j\sim q(.|\vec{\mathbf{x}}_i,i\in S_f)}^{k}\log(\frac{P_{q_2}(\vec{\mathbf{x}}_{j,f})p(i\in S_f)}{e^{s(\vec{\mathbf{x}}_j,\vec{\phi})}+P_{q_2}(\vec{\mathbf{x}}_{j,f}))}\Big)$$

$$=\frac{1}{|S^+|}\sum_{f=1}^{p}\sum_{i\in S_f^+}\Big(\log(\frac{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}}{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}+P_{q_2}(\vec{\mathbf{x}}_{i,f})})+\log(p(i\in S_f^+)^{k+1})$$

$$+\sum_{\vec{\mathbf{x}}_j\sim q(.|\vec{\mathbf{x}}_i,i\in S_f^+)}^{k}\log(\frac{P_{q_2}(\vec{\mathbf{x}}_{j,f})}{e^{s(\vec{\mathbf{x}}_j,\vec{\phi})}+P_{q_2}(\vec{\mathbf{x}}_{j,f})})\Big)$$

We now drop the term containing the probability of assignment to a feature type $p(i \in S_f^+)$ since it is outside of the learned model parameters $\vec{\phi}$ and fixed in advance:

$$\propto\frac{1}{|S^+|}\sum_{f=1}^{p}\sum_{i\in S_f^+}\Big(\log(\frac{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}}{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}+P_{q_2}(\vec{\mathbf{x}}_{i,f})})+\sum_{\vec{\mathbf{x}}_j\sim q(.|\vec{\mathbf{x}}_i,i\in S_f^+)}^{k}\log(\frac{P_{q_2}(\vec{\mathbf{x}}_{j,f})}{e^{s(\vec{\mathbf{x}}_j,\vec{\phi})}+P_{q_2}(\vec{\mathbf{x}}_{j,f})})\Big)$$

$$\xrightarrow[|S^+|\to\infty]{}\sum_{f=1}^{p}p(i\in S_f^+)E\Big[\log(\frac{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}}{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}+P_{q_2}(\vec{\mathbf{x}}_{i,f})})+\sum_{\vec{\mathbf{x}}_j\sim q(.|\vec{\mathbf{x}}_i,i\in S_f^+)}^{k}\log(\frac{P_{q_2}(\vec{\mathbf{x}}_{j,f})}{e^{s(\vec{\mathbf{x}}_j,\vec{\phi})}+P_{q_2}(\vec{\mathbf{x}}_{j,f})})\Big]$$

$$=\sum_{f=1}^{p}p(i\in S_f^+)E\Big[L(\vec{\mathbf{x}}|\vec{\phi},\text{target}=f)\Big]$$

Thus, the loss function is just a convex combination of the loss functions of the targeted classifiers for each of the p features, and by extension so is the gradient since:

$$\frac{\partial}{\partial\phi}\sum_{f=1}^{p}p(i\in S_f^+)E\Big[L(\vec{\mathbf{x}}|\vec{\phi},\text{target}=f)\Big]=\sum_{f=1}^{p}p(i\in S_f^+)\frac{\partial}{\partial\phi}E\Big[L(\vec{\mathbf{x}}|\vec{\phi},\text{target}=f)\Big]$$

Thus the algorithm will, at each step, learn a convex combination of the gradient for a targeted classifier on feature f, with weights proportional to the feature type sampling probabilities in step 1 of the sampling algorithm. □

A.2 Problems with Flattening Multinomial of Word2Vec

Write

References

Anonymized. Structured deep factorization machine: Towards general-purpose architectures. In *6th International Conference on Learning Representations*, Under review.

Chris Dyer. Notes on noise contrastive estimation and negative sampling. *arXiv preprint arXiv:1410.8251*, 2014.

Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 297–304, 2010.

Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pp. 2177–2185, 2014.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013b.

Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. In *In Proceedings of the International Conference on Machine Learning*, 2012.

Steffen Rendle. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pp. 995–1000. IEEE, 2010.

Tobias Schnabel, Igor Labutov, David M Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. In *EMNLP*, pp. 298–307, 2015.