FEAT2VEC: DENSE VECTOR REPRESENTATION FOR DATA WITH FEATURES

Anonymous authors
Paper under double-blind review

ABSTRACT

Methods that calculate dense vector representations 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 type of feature—that is, datasets that have arbitrary features beyond words. We study how to estimate dense 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 that all different feature types exist in a common space. We demonstrate our work on two datasets, and our experiments suggest that Feat2Vec significantly outperforms existing algorithms that do not leverage the structure of the data.

1 PRELIMINARIES

Informally, in machine learning a *dense representation*, or *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 features set—where each element 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: it does not optimize the embeddings of individual subfeatures, instead it reasons at a more abstract level of complexity. This enables making principled comparisons between different feature types.

2 FEAT2VEC

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 to how positive examples 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 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:

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

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. 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 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" or regular Factorization Machine 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 takes an additional input $\vec{\kappa}$, a vector that defines which features group together. Each entry i in $\vec{\kappa}$ is a vector of size d_i and it is used to represent the feature type $\mathcal{F}_i$ in the data. In a shallow model, a feature in the dataset interacts with all other features, but in the structured model it is the *groups* of feature types (as a whole) that interact with each other. For each feature type, the model applies a $d_i \times r$ feature extraction function ϕ_i that calculates an embedding for the feature type. More formally, this is:

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

$$\triangleq \sigma\big(s(\vec{x}, \vec{\phi})\big) \tag{7}$$

For notational convenience, we refer to this model as a scoring function $s(\cdot)$, and $\vec{x}_{\vec{\kappa}_i}$ as the vector that contains all of the subfeatures that belong to the group κ_i. A simple implementation for ϕ_i is a linear fully-connected layer, where the output of the r-th entry is:

$$\phi_i\big(\vec{x}_i, \vec{\beta}_i\big)_r = \sum_{a=1}^{d_i} \beta_{r_a} x_{i_a} \tag{8}$$

In Word2Vec, the embeddings for individual words of a document need to be similar. Feat2Vec has no such constraint. Individual subfeatures are not important; what matters is that feature types from within an observation are similar to each other.

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 7) requires positive and negative examples, we also need to supply observations that are not semantic. Consider a feature type $\mathcal{F}_i = \mathbb{R}^{d_i}$, that exists in very high dimensional space (i.e, 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, that has little theoretical guarantees (Dyer, 2014). In short, their approach samples a negative observation w from a noise distribution $\mathcal{Q}_{w2v}$, that is proportional to the empirical frequency of a word w in the training data.

Our 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 grouping subfeatures using the parameter κ in Equation 7, the model can reason on more complex entities. By entity, we mean a particular feature value. For example, in our experiments on a movie dataset, we use a "genre" feature type, where we group non-mutually exclusive indicators for movie genres including comedy, action, and drama films.

We start with a dataset S^+ of records with n 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: $\mathcal{Q}$

```
 1: function FEAT2VEC_SAMPLE($S^+, k, \alpha_1, \alpha_2$)
 2:     $S^- \leftarrow \emptyset$
 3:     for $\vec{x}^+ \in S^+$ do
 4:         Draw a random feature type $\kappa_i \sim \mathcal{Q}_1(\{\text{params}(\phi_i)\}_{i=1}^n, \alpha_1)$
 5:         for $j \in \{1, \dots, k\}$ do
 6:             $\vec{x}^- \leftarrow \vec{x}^+$                ▷ set initially to be equal to a positive sample
 7:             Draw a random subfeature $\vec{r} \sim \mathcal{Q}_2(\mathrm{X}_{\kappa_i}, \alpha_2)$
 8:             $\vec{x}^-_{\kappa_i} \leftarrow \vec{r}$          ▷ substitute the $i$-th feature type with the sampled one
 9:             $S^- \leftarrow S^- + \{\vec{x}^-\}$
10:         end for
11:     end for
12:     return $S^-$
13: end function
```

Explained in words, our negative sampling method iterates over all of the observations of the training dataset. For each observation $\vec{x}^+$, it randomly selects the i-th feature type from a noise distribution $\mathcal{Q}_1(\cdot)$. Then, it creates a negative observation that is identical to $\vec{x}^+$, except that its i-th feature type is replaced by a value sampled from a noise distribution $\mathcal{Q}_2(\cdot)$. 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.

We now describe the two noise distributions that we use. We use $p_{\mathcal{Q}}(x)$ to denote the probability of x under a distribution $\mathcal{Q}$.

Sampling Feature Types. The function params calculates the complexity of a feature extraction function ψ_i. To sample a feature type, we choose a feature type from a multinomial distribution with probabilities proportional a feature's complexity. This choice places more weight on features that have more parameters and thus are going to require more training iterations to properly learn. For categorical variables using a linear fully-connected layer, the complexity is simply proportional to the number of categories in the feature type:

$$P_{\mathcal{Q}_1}(\kappa_i | \, \text{params}(\phi_i)\}_{i=1}^n, \alpha_1 p) = \frac{\text{params}(\phi_i)^{\alpha_1}}{\sum_{k=1}^n \text{params}(\phi_k)^{\alpha_1}} = \frac{(r \times d_i)^{\alpha_1}}{\sum_{k=1}^n \text{params}(\phi_k)^{\alpha_1}}, \quad \alpha_1 \in [0,1] \tag{9}$$

However, if we have multiple intermediate layers for some feature extraction functions (e.g., convolutional layers), these parameters should also be counted. The hyper-parameter α_1 helps flatten the distribution. When $\alpha_1 = 0$, the feature types are sampled uniformly, and when $\alpha_1 = 1$, they are sampled proportional to their complexity. Figure A.1 in the Appendix provides a visualization of how the feature sampling rate varies with the hyperparameter for features with differing levels of complexity.

Sampling Subfeatures. To sample a subfeature value from within a feature type κ_i, we use a similar strategy to $\mathrm{Word2Vec}$ and use the subfeature empirical distribution:

$$P_{\mathcal{Q}_2}(x_j = x|\mathrm{X}_{\kappa_i}, \alpha_2) \;=\; \frac{\mathrm{count}(x)^{\alpha_2}}{\sum_{x'_{\kappa_i} \in S^+} \mathrm{count}(x'_{\kappa_i})^{\alpha_2}}, \quad \alpha_2 \in [0,1] \tag{10}$$

Here, $\mathrm{count}(x)$ is the number of times a subfeature x appeared in the training dataset S^+ .

This method will sometimes by chance generate negatively labeled samples that *do* exist in our sample of observed records. The literature offers two possibilities: in the Negative Sampling that $\mathrm{Word2Vec}$ follows, the duplicate negative samples are simply ignored (Dyer, 2014). Alternatively, it is possible to account for the possibility of random negative labels that appear identical to positively labeled data using Noise Contrastive Estimation (NCE) (Gutmann & Hyvärinen, 2010).

2.3 Learning From data

We optimize a NCE loss function, 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. However, an additional burden of NCE is that we need to calculate a partition function $Z_{\vec{x}}$ for each unique record type $\vec{x}$ that transform the score function $s(\cdot)$ into a well-behaved probability distribution that integrates to 1. Normally, this would introduce an astronomical amount of computation and greatly increase the complexity of the model. As a work-around, we appeal to the work of Mnih & Teh (2012), who showed that in the context of language models that setting the $Z_{\vec{x}} = 1$ in advance effectively does not change the performance of the model. The intuition is that if the underlying model has enough free parameters that it will effectively just learn the probabilities itself. Thus, it does not over/under predict the probabilities on average (since that will result in penalties on the loss function).

Written explicitly, the new structural probability model is:

$$\tilde{p}(Y = 1|\vec{\phi}, \vec{x}) = \frac{\exp\big(s(\vec{x}, \vec{\phi})\big)}{\exp(s\big(\vec{x}, \vec{\phi})\,\big) + P_{\mathcal{Q}}(\vec{x}|\alpha_1, \alpha_2)} \tag{11}$$

where $P_{\mathcal{Q}}(.)$ denotes the total probability of a record $\vec{x_i}$ being drawn from our negative sampling algorithm, conditional on the positively labeled record $\vec{x}^+$ the negative sample is drawn for:

$$P_{\mathcal{Q}}(\vec{x}|\alpha_1, \alpha_2, \mathrm{X}, \vec{x}^+) = P_{\mathcal{Q}_2}(\vec{x}_{\kappa_i}|\mathrm{X}_{\kappa_i}, \alpha_2) P_{\mathcal{Q}_1}(\kappa_i|\,\mathrm{params}(\phi_i)\}_{i=1}^n, \alpha_1) \tag{12}$$

Thus, our loss function L optimizes $\vec{\beta}$, the parameters of the feature extraction functions $\vec{\phi}$.

$$L(S) = \arg\min_{\beta} \frac{1}{|S^+|} \sum_{\vec{x}^+ \in S^+} \Big(\log(\tilde{p}(Y = 1|\vec{\phi}, \vec{x}^+)) \;+ \sum_{\vec{x}^- \sim \mathcal{Q}(\cdot|\vec{x}^+)}^{k} \log(\tilde{p}(Y = 0|\vec{\phi}, \vec{x}^-))\Big) \tag{13}$$

$\mathrm{Feat2Vec}$ has interesting theoretical properties. For example, it is well known that Factorization Machines can be used as a multi-label classifier: with at least two feature type, one can use one of the feature types as the target label, and the other type(s) as the input features to make a prediction. In such setting, the output indicates whether the label is associated with the input ($y = +1$), or not ($y = 0$), and therefore the input can be associated with more than one label. With n feature types, $\mathrm{Feat2Vec}$ is equivalent to optimizing a convex combination of the loss functions from n individual Factorization Machines. In other words, it optimizes n multi-label classifiers, where each classifier is optimized for a different target (i.e.,a specific feature type). We show the proof of this in the Appendix 1.

3 EMPIRICAL EVALUATION

3.1 DOES Feat2Vec ENABLE BETTER EMBEDDINGS?

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. In order to test the relative performance of our learned embeddings, we train our Feat2Vec algorithm and compare its performance in a targeted ranking task to Word2Vec. Thus, in our evaluation approach we use the embeddings of one entity, and evaluate its nearest-neighbors from a different entity from which we know the ground-truth. In particular, in the movie dataset we retrieve movie directors from actors embeddings; and from an educational dataset, we retrieve textbooks from user embeddings. We evaluate the rankings according to their mean percentile rank (MPR):

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

where R_i is the rank of the entity under our evaluation procedure for observation i. This measures on average how well we rank actual entities. 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.

3.1.1 DATASETS

Movies The Internet Movie Database (IMDB) is a publicly available dataset[1] of information related to films, television programs and video games. Though in this paper, we focus only on data on its 465,136 movies. Table 1 summarizes the feature types we use. It contains information on writers, directors, and principal cast members attached to each film, along with metadata.

Table 1: IMDB dataset

Feature Type Name	Type	# of feats.	Example for an instance
Runtime (minutes)	Real-valued	1	116
IMDB rating (0-10)	Real-valued	1	7.8
# of IMDB rating votes	Real-valued	1	435,682
Is adult film?	Boolean	2	False
Movie releaes year	Categorical	271	2001
Movie title	Text	165,471	"Ocean's", "Eleven"
Directors	Bag of categories	174,382	'Steven Soderbergh"
Genres	Bag of categories	28	"Crime", "Thriller"
Writers	Bag of categories	244,241	"George Johnson", "Jack Russell"
Principal cast members (actors)	Bag of categories	1,104,280	"George Clooney", "Brad Pitt", "Julia Roberts"

Education We use a dataset from an anonymized leading technology company that provides educational services. In this proprietary dataset, we have 57 million observations and 9 categorical feature types which include textbook identifier, user identifier, school identifier, and course the book is typically used with, along with other proprietary features. Here, each observation is an "interaction" a user had with a textbook.

3.1.2 FEATURE REPRESENTATION

Word2Vec For every observation in each of the datasets, we create a document that contains the same information that we feed into Feat2Vec. We prepend each feature by its feature type name, and we remove spaces from within features. In Figure 1 we show an example document. Some features may allow multiple values (e.g., multiple writers, directors). To feed these features into the models, for convenience, we constraint the number of values, by truncating each feature to no more than 10 levels (and sometimes less if reasonable). This results in retaining the full set of information for well over 95% of the values. We pad the sequences with a "null" category whenever necessary

[1]http://www.imdb.com/interfaces/

to maintain a fixed length. We do this consistently for both Word2Vec and Feat2Vec. 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.

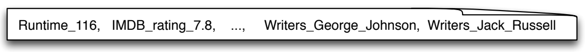
Runtime_116, IMDB_rating_7.8, ..., Writers_George_Johnson, Writers_Jack_Russell

Figure 1: Sample document for Word2Vec for the Ocean's Eleven movie

Feat2Vec Feature representation in Feat2Vec may require a feature extraction function for each feature type. Here, we explain how we build these functions:

- **Bag of categories, categorical, and boolean:** For all of the categorical variables, we learn a unique r-dimensional embedding for each entity using a linear fully-connected layer (Equation 8). We do not require one-hot encodings, and thus we allow multiple categories to be active; resulting in a single embedding for the group that is the sum of the embeddings of the subfeatures. This is ordering-invariant: the embedding of "Brad Pitt" would be the same when he appears in a movie as a principal cast member, regardless whether he was 1st or 2nd star. Though, if he were listed as a director it may result in a different embedding.
- **Text:** We preprocess the text by removing non alpha-numeric characters, stopwords, and stemming the remaining words. We then follow the same approach that we did for categorical variables. It would be easy to use more sophisticated methods (e.g, convolutions).
- **Real-valued:** 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 r, which we treat as the feature's embedding. Each intermediate layer has r 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 be unable to know that an IMDB rating of 5.5 is similar to 5.6.

3.2 Experimental Setup

Held-out set For our evaluation, we define a testing set that was not used to tune the parameters of the model. For the IMDB dataset, we select randomly a 10% sample of the observations that contain a director that appears at least twice in the database [2]. 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 the educational dataset, our testing set only has observations for textbooks and users that appear at least 10 times in training.

Hyper-parameters For both Feat2Vec and Word2Vec, we perform cross-validation on the loss function, by splitting the 10% of the training data randomly into a validation set, to determine the number of epochs to train, and then train the full training dataset with this number of epochs. [3] While regularization of the embeddings during training is possible, this did not dramatically change results, so we ignore this dimension of hyperparameters.

For training Feat2Vec we set $\alpha_1 = \alpha_2 = 3/4$ in the IMDB dataset; and $\alpha_1 = 0$ and $\alpha_2 = 0.5$ for the educational. In each setting, α_2 is set to the same flattening hyperparameter we use for Word2Vec to negatively sample words in a document. We learn $r = 50$ dimensional embeddings under both algorithms.

[2]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.

[3]Because we train Word2Vec with the `gensim` python library, it is impossible to recover the output weight matrix, and so the loss function is inaccessible for an outside document set. So, we created our own loss function that measures average within-document cosine similarity of all possible token pairs. The result was that both algorithms are trained for a similar number of epochs.

3.2.1 Results

After training, we use the cast members associated with the movies of the test set and attempt to predict the actual director the film was directed. 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. For the educational dataset, we simply use the user embedding directly to get the closest textbook.

Table 2 presents the results from our evaluation. Feat2Vec sizably outperforms Word2Vec in the MPR metric. Figure 2 shows the full distribution of rankings of the IMDB dataset, rather than summary statistics, in the form of a Cumulative Distribution Function (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. However, when 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 recommendation, it is the case that up until rank 8 or so, Feat2Vec outperforms Word2Vec still. 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 difficult. In fact, Feat2Vec predicts the actual director 2.43% of the times, while Word2Vec only does so 1.26% of the time, making our approach almost 2 times better in terms of Top-1 Precision metric.

Table 2: Mean percentile rank

Dataset	Feat2Vec	Word2Vec
IMDB	19.36%	24.15%
Educational	25.2%	29.2%

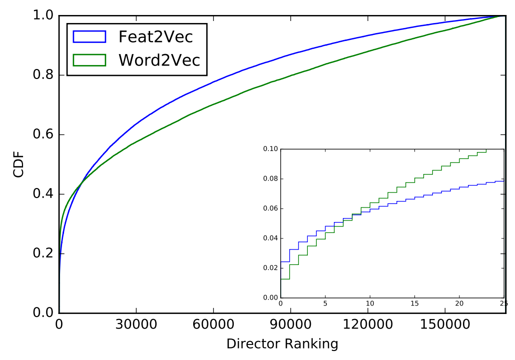

Figure 2: Cumulative Distribution Function of Director Rankings
(With Zoom-in to Top 25 Ranks)

3.3 Feat2Vec Performance with Continuous Inputs

We now focus on how well Feat2Vec performs on a real-valued feature with a complex feature extraction function. We evaluate the prediction of the real-valued rating of movies in the test dataset by choosing the IMDB rating embedding most similar[4] to the embedding of the movie's director, and compute the Root Mean Squared Error (RMSE) of the predicted rating in the test dataset. We also vary α_1, the flattening hyperparameter for feature sampling, to see what the effect this hyperparameter has on our performance. Intuitively, a low α_1 will greatly improve the quality of the ratings embeddings learned, since it has relatively few parameters and is otherwise sampled infrequently.

[4]As before, the metric is cosine similarity.

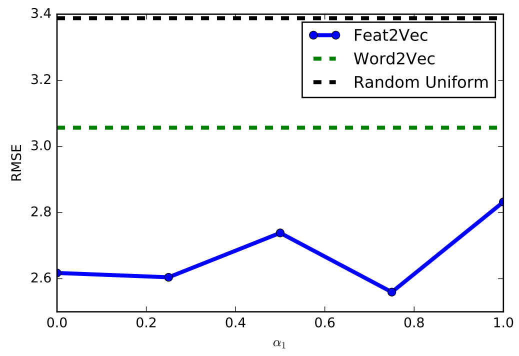

Figure 3: RMSE in Ratings Task as a Function of α_1

At the same time, with low α_1 the director feature will be sampled less since it is one of the most complex features to learn, so the learned director embeddings may be of poorer quality. Figure 3 displays the results of our experiment, benchmarked against Word2Vec's performance in the prediction task. We also show as a baseline the RMSE of a random uniform variable over the range of possible ratings (0 to 10). As is evident from the plot, Word2Vec performs a bit better than a random prediction, but is also handily outperformed by Feat2Vec across all hyper-parameter settings. The algorithm's performance does not seem very sensitive to the hyperparameter choice.

4 Relation to Prior Work

Algorithms that calculate continuous representations of entities other than words have been studied in the context of bioinformatics for biological sequences (Abrahamsson & Plotkin, 2009), or in machine translation for embeddings of complete sentences (Kiros et al., 2015). We recently discovered a promising direction for an algorithm still in development called StarSpace (Wu et al., 2017) with similar goals from ours. Even though they intend to be able to embed all types of features, at the time of the writing of this paper, their pre-print method was limited to only work for bag of words. While Feat2Vec can jointly learn embeddings for all feature values in a dataset, StarSpace samples a single arbitrary feature. Our preliminary experiments suggest that sampling a single feature does not produce embeddings that generalize well. Nonetheless, a limitation of our work is that we do not compare with StarSpace, which future work may decide to do.

5 Conclusion

Motivated by the success of Word2Vec, this paper proposes a novel algorithm that relies on Structured Deep Factorization Machines with implicit sampling to learn general purpose embedding representations of different types of features. Future work could study how to reduce the amount of human knowledge our approach requires; for example by automatically grouping features into entities, or by automatically choosing a feature extraction function. These ideas can extend to our codebase that we make available [5].

Our evaluation is limited to two datasets—a public dataset that is freely available, and a proprietary one that we are unable to distribute. Though further experimentation is definitely necessary, we believe that our results are extremely encouraging. Feat2Vec's embeddings are able to capture relationships across features that can be twice as better as Word2Vec on some evaluation metrics.

Feat2Vec exploits the structure of a datasets to learn embeddings in a way that is structurally more sensible than existing methods. The sampling method, and loss function that we use have interesting theoretical properties. To the extent of our knowledge, Feat2Vec is the first method able to calculate continuous representations of data with arbitrary feature types.

[5]The code for the Feat2Vec algorithm is available here and the empirical experiments for the IMDB data can be found here

REFERENCES

Erik Abrahamsson and Steven S Plotkin. Biovec: a program for biomolecule visualization with ellipsoidal coarse-graining. *Journal of Molecular Graphics and Modelling*, 28(2):140–145, 2009.

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.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In *Advances in neural information processing systems*, pp. 3294–3302, 2015.

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.

Ledell Wu, Adam Fisch, Sumit Chopra, Keith Adams, Antoine Bordes, and Jason Weston. Starspace: Embed all the things! *arXiv preprint arXiv:1709.03856*, 2017.

A Appendixes

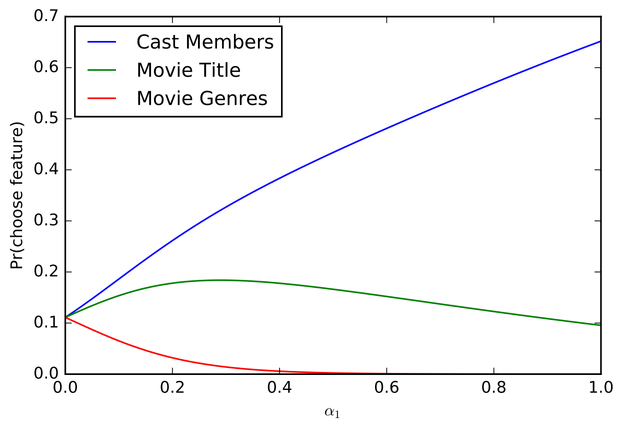

Figure A.1: Feature Sampling Probabilities as a Function of α_1

A.1 Proof to Theorem 1

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

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

$$\begin{aligned}
L(S^+|\vec{\phi}) =& \frac{1}{|S^+|}\sum_{\vec{x}^+\in S^+}\Big(\log(\tilde{p}(Y=1|\vec{\phi},\vec{x}^+)) + \sum_{\vec{x}^-\sim\mathcal{Q}(.|\vec{x}^+)}^{k}\log(\tilde{p}(Y=0|\vec{\phi},\vec{x}^-))\Big)\\
=&\frac{1}{|S^+|}\sum_{\vec{x}^+\in S^+}\Big(\log(\tilde{p}(Y=1|\vec{\phi},\vec{x}^+,\vec{x}^+\in S^+_{\kappa_i})p(\vec{x}^+\in S^+_{\kappa_i}))\\
&+\sum_{\vec{x}^-\sim\mathcal{Q}(.|\vec{x}^+)}^{k}\log(\tilde{p}(Y=0|\vec{\phi},\vec{x}^-,\vec{x}^+\in S^+_{\kappa_i})p(\vec{x}^+\in S^+_{\kappa_i}))\Big)\\
=&\frac{1}{|S^+|}\sum_{i=1}^{p}\sum_{\vec{x}^+\in S^+_{\kappa_i}}\Big(\log(\frac{e^{s(\vec{x}^+,\vec{\phi})}p(\vec{x}^+\in S^+_{\kappa_i})}{e^{s(\vec{x}^+,\vec{\phi})}+P_{\mathcal{Q}}(\vec{x}^+|\vec{x}^+,\vec{x}^+\in S^+_{\kappa_i})})\\
&+\sum_{\vec{x}^-\sim\mathcal{Q}(.|\vec{x}^+,\vec{x}^+\in S^+_{\kappa_i})}^{k}\log(\frac{P_{\mathcal{Q}}(\vec{x}^-|\vec{x}^+,\vec{x}^+\in S^+_{\kappa_i})p(\vec{x}^+\in S^+_{\kappa_i})}{e^{s(\vec{x}^-,\vec{\phi})}+P_{\mathcal{Q}}(\vec{x}^-|\vec{x}^+,\vec{x}^+\in S^+_{\kappa_i})})\Big)
\end{aligned}$$

Note now that $P_{\mathcal{Q}}(\vec{x}|\vec{x}^+, \vec{x}^+ \in S^+_{\kappa_i})$ is simply the probability of the record's feature value $\vec{x}_f$ under the second step noise distribution $\mathcal{Q}_2(\mathrm{X_f}, \alpha_2)$: $P_{\mathcal{Q}}(\vec{x}|\vec{x}^+, \vec{x}^+ \in S^+_{\kappa_i}) = P_{\mathcal{Q}_2}(\vec{x}_f)$

$$\begin{aligned}
=&\frac{1}{|S^+|}\sum_{i=1}^{p}\sum_{\vec{x}^+\in S^+_{\kappa_i}}\Big(\log\big(\frac{e^{s(\vec{x}^+,\vec{\phi})}p(\vec{x}^+\in S^+_{\kappa_i})}{e^{s(\vec{x}^+,\vec{\phi})}+P_{\mathcal{Q}_2}(\vec{x}^+_{\kappa_i})}\big)+\sum_{\vec{x}^-\sim\mathcal{Q}(.|\vec{x}^+,i\in S^+_{\kappa_i})}^{k}\log\big(\frac{P_{\mathcal{Q}_2}(\vec{x}^-_f)p(\vec{x}^+\in S^+_{\kappa_i})}{e^{s(\vec{x}^-,\vec{\phi})}+P_{\mathcal{Q}_2}(\vec{x}^-_f))}\Big)\\
=&\frac{1}{|S^+|}\sum_{i=1}^{p}\sum_{\vec{x}^+\in S^+_{\kappa_i}}\Big(\log\big(\frac{e^{s(\vec{x}^+,\vec{\phi})}}{e^{s(\vec{x}^+,\vec{\phi})}+P_{\mathcal{Q}_2}(\vec{x}^+_{\kappa_i})}\big)+\log(p(\vec{x}^+\in S^+_{\kappa_i})^{k+1})\\
&+\sum_{\vec{x}^-\sim\mathcal{Q}(.|\vec{x}^+,\vec{x}^+\in S^+_{\kappa_i})}^{k}\log\big(\frac{P_{\mathcal{Q}_2}(\vec{x}^-_f)}{e^{s(\vec{x}^-,\vec{\phi})}+P_{\mathcal{Q}_2}(\vec{x}^-_f)}\big)\Big)
\end{aligned}$$

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

$$\begin{aligned}
\propto&\frac{1}{|S^+|}\sum_{i=1}^{p}\sum_{\vec{x}^+\in S^+_{\kappa_i}}\Big(\log\big(\frac{e^{s(\vec{x}^+,\vec{\phi})}}{e^{s(\vec{x}^+,\vec{\phi})}+P_{\mathcal{Q}_2}(\vec{x}^+_{\kappa_i})}\big)+\sum_{\vec{x}^-\sim\mathcal{Q}(.|\vec{x}^+,\vec{x}^+\in S^+_{\kappa_i})}^{k}\log\big(\frac{P_{\mathcal{Q}_2}(\vec{x}^-_f)}{e^{s(\vec{x}^-,\vec{\phi})}+P_{\mathcal{Q}_2}(\vec{x}^-_f)}\big)\Big)\\
\xrightarrow[|S^+|\to\infty]{}&\sum_{i=1}^{p}p(\vec{x}^+\in S^+_{\kappa_i})E\Big[\log\big(\frac{e^{s(\vec{x}^+,\vec{\phi})}}{e^{s(\vec{x}^+,\vec{\phi})}+P_{\mathcal{Q}_2}(\vec{x}^+_{\kappa_i})}\big)+\sum_{\vec{x}^-\sim\mathcal{Q}(.|\vec{x}^+,\vec{x}^+\in S^+_{\kappa_i})}^{k}\log\big(\frac{P_{\mathcal{Q}_2}(\vec{x}^-_f)}{e^{s(\vec{x}^-,\vec{\phi})}+P_{\mathcal{Q}_2}(\vec{x}^-_f)}\big)\Big]\\
=&\sum_{i=1}^{p}p(\vec{x}^+\in S^+_{\kappa_i})E\Big[L(\vec{x}|\vec{\phi},\text{target} = f)\Big]
\end{aligned}$$

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:

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

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. $\square$