Feat2Vec: Supervised and Self-Supervised Embeddings of Data

David S. Hippocampus Department of Computer Science
Cranberry-Lemon University
Pittsburgh, PA 15213
hippo@cs.cranberry-lemon.edu

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. We study how to estimate dense representations when multiple feature types exist within a dataset, both for supervised learning where explicit labels are available, and self-supervised learning where there are no labels. Feat2Vec calculates embeddings for data with multiple feature types enforcing that all different feature types exist in a common space. In the supervised case, we show that our method has advantages over recently proposed methods; such as enabling higher prediction accuracy, and providing a way to avoid the cold-start problem. In the self-supervised case, our experiments suggest that Feat2Vec better compresses data. We believe that we are the first to propose a method for learning self-supervised embeddings that leverage the structure of multiple feature types.

Introduction

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. In general, we consider two kind of models that produce embeddings: **(i) supervised methods**, like matrix factorization, calculate embeddings that are highly tuned to a prediction task. For example, in the Nextflix challenge, movie identifiers are embedded to predict user ratings. On the other hand, **(ii) self-supervised methods** (sometimes less precisely referred as unsupervised methods) like the Word2Vec models (Mikolov et al., 2013), are often used for transfer learning. In this context, transfer learning uses self-supervised embeddings for predicting an unrelated task. For example, Word2Vec embeddings may be used for analogy solving, or for sentiment analysis (Le and Mikolov, 2014).

In this paper we propose Feat2Vec as a novel method that embeds arbitrary feature types, such as text, numerical or categorical data, in a common vector space—for both supervised and self-supervised scenarios. The contributions of this paper are two-fold. First, Feat2Vec is the first algorithm that is able to calculate general-purpose embeddings with self-supervision that are useful for transfer learning. Second, we allow calculating embeddings of arbitrary features. The advantage of arbitrary features in the self-supervised scenario is that it a more general approach. Consider the non-trivial work that an engineer needs to do to extend a model like Word2Vec to support additional features. The authors of the seminal Doc2Vec (Le and Mikolov, 2014) paper needed to design both a new neural network and a new sampling strategy to simply add a single feature (Document ID). In the supervised scenario, the advantage of arbitrary features is that our experiments show that the cold-start problem can be avoided by instead of embedding a single categorical identifier, we can embed a description of the item with words.

Preliminaries

Factorization Machine (Rendle, 2010) is one of the most successful methods for general-purpose factorization. Consider a degree-2 polynomial (quadratic) regression, where we want to predict a target variable y from a vector of inputs $\vec{x} \in \mathbb{R}^n$:

$$\hat{y}(\vec{x}; \vec{b}, \vec{w}) = \omega\big(b_0 + \sum_i b_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} w_{i,j}\ x_i x_j\big) \quad (1)$$

In words, n is the total number of features, the term b_0 is an intercept, b_i is the strength of the i-th feature, and $w_{i,j}$ is the interaction coefficient between the i-th and j-th feature. ω is an activation function.

Factorization Machine replaces the two-way individual pairwise parameters $w_{i,j}$ for each interaction with a vector of parameters $\vec{w}_i$ for each feature. This is a rank-r vector of latent factors—*embeddings* in the neural literature—that encode the interaction between features and replaces the quadratic regression model with the following:

$$\hat{y}(\vec{x}; \vec{b}, \vec{w}) = \omega\big(b_0 + \sum_i b_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} (x_i \vec{w}_i) \cdot (x_j \vec{w}_j)\big) \quad (2)$$

Intuitively, the dot product $(\cdot)$ returns a scalar that measures the (dis)similarity between the latent factors of features x_i and x_j. Polynomial regression has n^2 interaction parameters, and Factorization Machine has $n \times r$. While setting $r \ll n$ makes the model less expressive, factorization will typically exploit features having some shared latent structure. Factorization Machine may dramatically reduce the number of parameters to estimate. Rendle (2010) shows that when the feature vector $\mathbf{x}$ consists only of two categorical features in one-hot encoding, Factorization Machine is equivalent to the popular Matrix Factorization algorithm (Koren, Bell, and Volinsky, 2009).

Feat2Vec

We now describe how Feat2Vec extends the Factorization Machine model by allowing grouping of features, and enabling arbitrary feature extraction functions . We also report a supervised method to learning Feat2Vec, as well as a novel self-supervised training procedure.

Model

We propose a framework for extending factorization machine with neural methods, by introducing structure into the feature interactions. Specifically, we do this by defining *feature groups*, $\vec{\kappa}$, where each group contains features of a particular type. Explicitly, $\vec{\kappa}$ is a partition of the set of feature columns in a dataset and each set within the partition is a feature group. For example, the 256 pixel intensities from a 16×16 image may be grouped into an "image" feature group. The embeddings of a feature group are then learned via a *feature extraction function*, ϕ_i, defined for each feature group. Feat2Vec will then extract features from each feature group, and build r latent factors from them. In Factorization Machine, all the feature embeddings interact with each other, while in Feat2Vec, the interactions only occur between different feature *groups*.

Formally, the addition of deep extraction methods yields the following statistical model:

$$\hat{y}(\vec{x}, \vec{b}, \vec{\phi}) = \omega\left(b_0 + \sum_{i=1}^{n} b_i x_i \; + \; \sum_{i=1}^{|\vec{\kappa}|} \sum_{j=i}^{|\vec{\kappa}|} \phi_i(\vec{x}_{\vec{\kappa}_i}) \cdot \phi_j(\vec{x}_{\vec{\kappa}_j})\right) \tag{3}$$

In this notation, $\vec{x}_{\vec{\kappa}_i}$ is a subvector that contains all of the features that belong to the group $\vec{\kappa}_i$. ϕ_i is a feature extraction that inputs the i-th feature group of the instance, and returns an r-dimensional embedding. The feature extraction function ϕ_i can allow for an arbitrary processing of its subfeatures $\vec{x}_{\vec{\kappa}_i}$. Instead of individual sub-features interacting among each other, as in Factorization Machine, the embeddings of feature groups interact with those of other groups. The intuition is that by grouping (sub-)features as a single entity, we can can reason on a higher level of abstraction.

As a concrete example of an application of this grouping/feature extraction, we might group the individual words of a document into a "document" feature group, and allow this document embedding to then interact with learned embeddings of other document metadata (such as author id). We might expect the extraction function ϕ for the words in a document to extract features that characterize the attributes of the document taken as a whole, rather than simply the sum of its individual words.

Figure 1 compares existing factorization methods with our novel model. In this example, Feat2Vec is using two feature groups: the first group only has a single feature which is projected to an embedding (just like a regular Factorization Machine); the second group has multiple features, which are together projected to a single embedding.

The simplest implementation for ϕ_i is a linear fully-connected layer, where the output of the r-th entry is:

$$\phi_i\big(\vec{x}_i; \vec{w}\big)_r = \sum_{a=1}^{d_i} w_{r_a} x_{i_a} \tag{4}$$

Note that without loss of generality, we could define a model that is equivalent to a Factorization Machine by requiring each feature group to be a singleton : $\vec{\kappa} = \{\{x_1\}, \{x_2\} \ldots \{x_n\}\}$ and the linear extraction function presented in Equation 4.

We can use Feat2Vec to overcome the cold-start problem. This is only possible when there is an alternative description of the item available (for example an image or a passage of text). We address this problem by treating the words as indexed features, but placed within a structured feature group κ_w. A feature extraction function ϕ acts on the features in κ_w, and the other features interact with the words only via the output of ϕ.

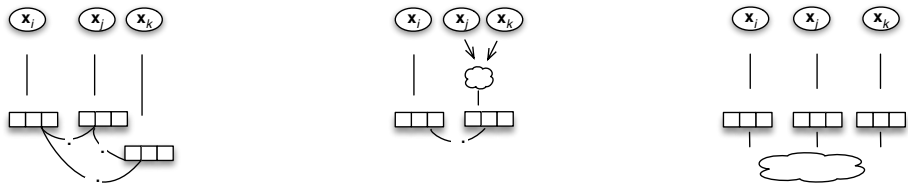
Figure 1c shows an approach of using neural networks within factorization machines that has been proposed multiple times (Dziugaite and Roy, 2015; Guo et al., 2017). It replaces the dot product of factors with a learned neural function, which has been shown to improve predictive accuracy for various tasks. In this case, fast inference for cold-start documents using pre-computed label embeddings is no longer possible. It needs to store the entire neural function that takes the embeddings as inputs. Another shortcoming of replacing the dot product with a neural function is that it would no longer be possible to interpret the embeddings as containing latent factors related to the target task; There may be highly complex mappings from the embeddings to the final output via this neural function.

Supervised Learning from Data

We can learn the the parameters of a deep factorization model $\boldsymbol{\theta}$ using training data by minimizing a loss function $\mathcal{L}$:

$$\arg\min_{\vec{\theta}} \sum_{x} \mathcal{L}\big(y(x), \hat{y}(x; \vec{\theta})\big) + \gamma ||\boldsymbol{\theta}||^w \tag{5}$$

Here, $y(x)$ is the true target value for x obtained from training data, and $\hat{y}(x)$ is the one estimated by the model



(a) **Shallow** (b) **Feat2Vec** (c) **"Neural"**

Figure 1: **Network architectures for factorization models**. The white clouds (☁) represent deep layers, for example a convolutional network for text features, while the dots ($\cdot$) denote dot products.

(Equation 3); $\boldsymbol{\theta}$ represents the parameters learned in during training (i.e. the b_i terms and parameters associated with the extraction functions ϕ_i in Equation 3). The hyperparameter γ controls the amount of regularization. For labeling and classification tasks, we optimize the binary cross-entropy.

It is straightforward to optimize Equation 5 directly. In the multi-class scenario, if the number of labels is very large, it is common practice use a binary classifier with implicit sampling (Dyer, 2014). In this case, we would have at least two feature groups—one of the feature groups is the target label that we want to predict, such as rating, and the other group(s) is the input from which we want to make the prediction, such as review text. The output indicates whether the label is associated with the input ($y = 1$), or not ($y = 0$). The datasets we use for our labeling experiments only contains positive labels, thus for each training example we sample a set of negative labels equal to the number of positive labels. It is typical to use one of the following sampling strategies according to the best validation error, in each case excluding the actual positive labels for each training example – (i) uniformly from all possible labels, or (ii) from the empirical distributions of positive labels (Rendle et al., 2009; Rendle and Freudenthaler, 2014).

Self-supervised Learning From Data

We now discuss how Feat2Vec can be used to learn embeddings in an self-supervised setting with no explicit target for prediction.

The training dataset for a Feat2Vec model consists of only the observed data. In natural language, these would be documents written by humans, along with document metadata. Since Feat2Vec (Equation 3) requires positive and negative examples, we also need to supply unobserved data as negative examples. Consider a feature group $\vec{\kappa}_i$ that exists in very high dimensional space. This could be a one-hot encoding of a categorical variable with a large number of possible values. In such a scenario, it is overwhelmingly costly to feed the model all negative labels, particularly if the model is sparse.

A shortcut around this is *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. For example, Word2Vec samples a negative observation from a noise distribution $\mathcal{Q}_{w2v}$, that is proportional to the empirical frequency of a word in the training data.

We introduce a new implicit sampling method that enables learning self-supervised embeddings for feature groups. We can learn the correlation of feature groups within a dataset by imputing negative labels, simply by generating unobserved records as our negative samples. Unlike Word2Vec, we do not constraint features types to be words. By grouping subfeatures using the parameter κ in Equation 3, the model can reason on more abstract entities in the data. By entity, we mean a particular feature group value. For example, in our experiments on a movie dataset, we use a "genre" feature group, 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 $|\vec{\kappa}|$ feature groups. We then mark all observed records in the training set as positive examples. For each positive record, we generate k negative labels using the 2-step algorithm documented in Algorithm 1.

Algorithm 1 Implicit sampling algorithm for self-supervised Feat2Vec

function FEAT2VEC_SAMPLE($S^+, k, \alpha_1, \alpha_2$)
  $S^- \leftarrow \emptyset$
  for $\vec{x}^+ \in S^+$ **do**
    Draw a random feature group $\kappa_i \sim \mathcal{Q}_1(\{\text{params}(\phi_i)\}_{i=1}^{|\vec{\kappa}|}, \alpha_1)$
    for $j \in \{1, \ldots, k\}$ **do**
      $\vec{x}^- \leftarrow \vec{x}^+$ ▷ set initially to be equal to the positive sample
      Draw a random feature group value $\tilde{x} \sim \mathcal{Q}_2(\mathrm{X}_{\kappa_i}, \alpha_2)$
      $\vec{x}^-_{\kappa_i} \leftarrow \tilde{x}$ ▷ substitute the i-th feature type with the sampled one
      $S^- \leftarrow S^- + \{\vec{x}^-\}$
    end for
  end for
  return S^-
end function

Explained in words, our negative sampling method for self-supervised learning iterates over all of the observations of the training dataset. For each observation $\vec{x}^+$, it randomly selects the i-th feature group 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 group 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 be the case. We now describe the two noise distributions that we use. Let $P_{\mathcal{Q}}(x)$ denote the probability of x under a distribution $\mathcal{Q}$.

Sampling Feature Groups. The function params calculates the complexity of a feature extraction function ϕ_i. To sample a feature group, we choose a feature group κ_i from a multinomial distribution with probabilities proportional a feature's complexity. By complexity, we mean the number of parameters we learn that are associated with a particular feature group. This choice places more weight on features that have more parameters and thus are going to require more training iterations to properly learn. The sampling probabilities of each feature group are:

$$P_{\mathcal{Q}_1}(\kappa_i|\,\text{params}(\phi_i)\}_{i=1}^{|\vec{\kappa}|},\alpha_1) \;=\frac{\text{params}(\phi_i)^{\alpha_1}}{\sum_{j=1}^{|\vec{\kappa}|}\text{params}(\phi_j)^{\alpha_1}} \tag{6}$$

For categorical variables using a linear fully-connected layer, the complexity is simply proportional to the number of categories in the feature group. However, if we have multiple intermediate layers for some feature extraction functions (e.g., convolutional layers), these parameters should also be counted towards a feature group's complexity. The hyper-parameter $\alpha_1 \in [0,1]$ helps flatten the distribution. When $\alpha_1 = 0$, the feature groups are sampled uniformly, and when $\alpha_1 = 1$, they are sampled exactly proportional to their complexity.

Sampling Feature Group Values. To sample an entity within a feature group κ_i, we use a similar strategy to Word2Vec and use the empirical distribution of values:

$$P_{\mathcal{Q}_2}(x|\mathrm{X}_{\kappa_i},\alpha_2) \;=\frac{\text{count}(x)^{\alpha_2}}{\sum_{x'_{\kappa_i}\in S^+}\text{count}(x'_{\kappa_i})^{\alpha_2}},\quad \alpha_2\in[0,1] \tag{7}$$

Here, count(x) is the number of times a feature group value x appeared in the training dataset S^+ , and α_2 is again a flattening hyperparameter.

This method will sometimes by chance generate negatively labeled samples that *do* exist in our sample of observed records. The literature offers two solutions: in the Negative Sampling of Word2Vec, duplicate negative samples are ignored (Dyer, 2014). Instead, we account for the probability of random negative labels being identical to positively labeled data using Noise Contrastive Estimation (NCE) (Gutmann and Hyvärinen, 2010).

The Loss Function for Self-Supervised Learning For our self-supervised learning of embeddings, we optimize a NCE loss function, to adjust the structural statistical model $\hat{y} = p(y=1|\vec{x},\vec{\phi},\boldsymbol{\theta})$ expressed in Equation 3, to account for the possibility of random negative labels that appear identical to positively labeled data. Since in self-supervised learning we only deal with a dichotomous label, indicating a positive or negative sample, we restrict our attention to usage of Equation 3 with ω as a logistic link function.

An additional burden of NCE is that we need to calculate a partition function $Z_{\vec{x}}$ for each unique record type $\vec{x}$ in the data that transforms the probability $\hat{y}$ of a positive or negative label into a well-behaved distribution that integrates to 1. Normally, this would introduce an astronomical amount of computation and greatly increase the complexity of the model. Instead, we appeal to the work of Mnih and Teh (2012), who show that in the context of language models 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 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:

$$\tilde{p}(y=1|\vec{x},\vec{\phi},\boldsymbol{\theta}) = \frac{\exp\big(s(\vec{x},\vec{\phi},\boldsymbol{\theta})\big)}{\exp(s\big(\vec{x},\vec{\phi},\boldsymbol{\theta})\big)+P_{\mathcal{Q}}(\vec{x}|\alpha_1,\alpha_2)} \tag{8}$$

$$s(\vec{x},\vec{\phi},\boldsymbol{\theta}) = b_0+\sum_{i=1}^{n}b_i x_i \;+\sum_{i=1}^{|\vec{\kappa}|}\sum_{j=i}^{|\vec{\kappa}|}\phi_i(\vec{x}_{\vec{\kappa}_i})\cdot\phi_j(\vec{x}_{\vec{\kappa}_j}) \tag{9}$$

$s(.)$ denotes the score of a record $\vec{x}$ given parameters/extraction functions, and $P_{\mathcal{Q}}(.)$ denotes the 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:

$$P_{\mathcal{Q}}(\vec{x}|\alpha_1,\alpha_2,\mathrm{X},\vec{x}^+) = P_{\mathcal{Q}_2}(\vec{x}_{\kappa_i}|\mathrm{X}_{\kappa_i},\alpha_2)P_{\mathcal{Q}_1}(\kappa_i|\,\text{params}(\phi_i)\}_{i=1}^{n},\alpha_1) \tag{10}$$

Our loss function L optimizes $\boldsymbol{\theta}$, the parameters of the feature extraction functions $\vec{\phi}$, while accounting for the probability of negative samples.

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

Feat2Vec has interesting theoretical properties. It is well known that Factorization Machines can be used as a multi-label classifier: with at least two features, one can use one feature as a target label, and the other(s) as the input feature 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, self-supervised Feat2Vec can be shown to be equivalent to optimizing a convex combination of the loss functions from n Factorization Machines. In

Table 1: Supervised Yelp rating prediction

	MSE	Improvement over MF
Feat2Vec	**0.480**	**69.2** %
DeepCoNN	1.441	19.6 %
MF (Matrix Factorization)	1.561	-

other words, it optimizes n multi-label classifiers, where each classifier is optimized for a different target (i.e.,a specific feature group). We show the proof of this in the Supplemental Materials **??**.

Empirical Results

Supervised Embeddings

We now address our working hypotheses for evaluating supervised embeddings. For all our experiments we define a development set and a single test set which is 10% of the dataset, and a part of the development set is used for early stopping or validating hyper-parameters. Since we only observe positive labels, for each positive label in the test set we sample negative labels according to the label frequency. This ensures that if a model merely predicts the labels according to their popularity, it would have an AUC of 0.5. For the regression task, we use mean squared error (MSE) as the evaluation metric. In preliminary experiments we noticed that regularization slows down convergence with no gains in prediction accuracy, so we avoid overfitting only by using early stopping. We share most of the code for the experiments online[1] for reproducibility.

For our feature extraction function ϕ for text, we use a Convolutional Neural Network (CNN) that has been shown to be effective for natural language tasks (Kalchbrenner, Grefenstette, and Blunsom, 2014; Weston, Chopra, and Adams, 2014). Instead of tuning the hyper-parameters, we follow previously published guidelines (Zhang and Wallace, 2015).

Comparison with alternative CNN-based text factorization We now compare with a method called DeepCoNN, a deep network specifically designed for incorporating text into matrix factorization (Zheng, Noroozi, and Yu, 2017)—which reportedly, is the state of the art for predicting customer ratings when textual reviews are available. To make results comparables, the Feat2Vec experiments use the same feature extraction function used by DeepCoNN. We evaluate on the Yelp dataset[2], which consists of 4.7 million reviews of restaurants. For each user-item pair, DeepCoNN concatenates the text from all reviews for that item and all reviews by that user. The concatenated text is fed into a feature extraction function followed by a factorization machine. In contrast, for Feat2Vec, we build 3 feature groups: item identifiers (in this case, restaurants), users and review text.

Table 1 compares our methods to DeepCoNN's published results because a public implementation is not available. We see that Feat2Vec provides a large performance increase when comparing the reported improvement, over Matrix Factorization, of the mean squared error. Our approach is more general, and we claim that it is also more efficient. Since DeepCoNN concatenates text, when the average reviews per user is $\bar{n_u}$ and reviews per item is $\bar{n_i}$, each text is duplicated on average $\bar{n_i} \times \bar{n_u}$ times per training epoch. In contrast, for Feat2Vec each review is seen only once per epoch. Thus it can be 1-2 orders of magnitude more efficient for datasets where $\bar{n_i} \times \bar{n_u}$ is large.

Self-Supervised Embeddings

Does Feat2Vec **enable better embeddings?** Ex ante, it is unclear to us how to evaluate the performance of an self-supervised embedding algorithm, but we felt that a reasonable task would be a ranking task one might practically attempt using our datasets. This task will 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 self-supervised Feat2Vec algorithm and compare its performance in a targeted ranking task to Word2Vec's CBOW algorithm for learning embeddings. In our evaluation approach, we compare the cosine similarity of the embeddings of two entities where these entities are known to be associated with each other since they appear in the same observation in a test dataset. 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.

Datasets

Movies The Internet Movie Database (IMDB) is a publicly available dataset[3] of information related to films, television programs and video games. Though in this paper, we focus only on data on its 465,136 movies. The dataset contains information on writers, directors, and principal cast members attached to each film, along with metadata.

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, along with other proprietary features. Each observation is an "interaction" a user had with a textbook.

[1] `https://goo.gl/zEQBiA`

[2] https://www.yelp.com/dataset/challenge

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

Yelp We use the Yelp dataset from our supervised experiments to evaluate the efficacy of self-supervised embeddings in ratings prediction.

Results After training IMDB embeddings, we use the cast members associated with movies in 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 most similar textbooks.

Table 2 presents the results from our evaluation. Feat2Vec sizably outperforms CBOW in the MPR metric. In fact, Feat2Vec predicts the actual director 2.43% of the times, while CBOW 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	CBOW
IMDB	19.36%	24.15%
Educational	25.2%	29.2%

Self-Supervised Feat2Vec Performance with Continuous Inputs

We now focus on how well Feat2Vec performs on a real-valued feature. We expect this task to highlight Feat2Vec's advantage over token-based embedding learning algorithms, such as Word2Vec, since our rating embedding extraction function (a 3-layer DNN) will require embeddings of numerically similar ratings to be close , while Word2Vec will treat two differing ratings tokens as completely different entities. 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. Feat2Vec scores an RMSE of 2.6, while Word2Vec scores 3.0.

We also use the Yelp dataset, predicting the most similar rating embedding to the review text embedding produced by Feat2Vec, Word2Vec (CBOW), and Doc2Vec (DM). For Word2Vec and Doc2Vec, the review text embedding is the average of word embeddings, analogous to the context vector used for learning in these algorithms. Table 3 reports the results. As a baseline, we compare our metrics to random predictions using the empirical distribution of ratings in the data. Feat2Vec and Doc2Vec yield comparable MSE, but Feat2Vec outperforms Doc2Vec and Word2Vec by a substantial margin in error rate. In fact, Doc2Vec does no better than a random prediction in terms of error rate. Only Feat2Vec performs well along both metrics.

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

Table 3: Self-Supervised Yelp rating prediction

	MSE	Error Rate
Random Rating	3.65	73%
Feat2Vec	2.94	**55**%
Word2Vec	3.42	61%
Doc2Vec	**2.92**	73%

Conclusion

Embeddings have proven useful in a wide variety of contexts, but they are typically built from datasets with a single feature type as in the case of Word2Vec, or tuned for a single prediction task as in the case of Factorization Machine. We believe Feat2Vec is an important step towards general-purpose methods, because it decouples feature extraction from prediction for datasets with multiple feature types, it can be self-supervised, and its embeddings are easily interpretable.

In the supervised setting, Feat2Vec is able to calculate embeddings for passages of texts. We show results outperforming an algorithm specifically designed for text—even when using the same feature extraction CNN.

In the self-supervised setting, Feat2Vec exploits the structure of a dataset to learn embeddings in a way that is structurally more sensible than existing methods. This yields performance improvements in our ranking and prediction tasks. The sampling method, and loss function that we use have interesting theoretical properties. To the extent of our knowledge, Self-Supervised Feat2Vec is the first method able to calculate continuous representations of data with arbitrary feature types.

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]. Though further experimentation is necessary, we believe that our results are an encouraging step towards general-purpose embedding models.

References

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

Dziugaite, G. K., and Roy, D. M. 2015. Neural network matrix factorization. *CoRR* abs/1511.06443.

Guo, H.; Tang, R.; Ye, Y.; Li, Z.; and He, X. 2017. Deepfm: A factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*.

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

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

Kalchbrenner, N.; Grefenstette, E.; and Blunsom, P. 2014. A convolutional neural network for modelling sentences. *CoRR* abs/1404.2188.

Koren, Y.; Bell, R.; and Volinsky, C. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42(8):30–37.

Le, Q., and Mikolov, T. 2014. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 1188–1196.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 3111–3119.

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

Rendle, S., and Freudenthaler, C. 2014. Improving pairwise learning for item recommendation from implicit feedback. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, 273–282. New York, NY, USA: ACM.

Rendle, S.; Freudenthaler, C.; Gantner, Z.; and Schmidt-Thieme, L. 2009. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, 452–461. Arlington, Virginia, United States: AUAI Press.

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

Weston, J.; Chopra, S.; and Adams, K. 2014. #tagspace: Semantic embeddings from hashtags. In Moschitti, A.; Pang, B.; and Daelemans, W., eds., *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, 1822–1827. ACL.

Zhang, Y., and Wallace, B. 2015. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*.

Zheng, L.; Noroozi, V.; and Yu, P. S. 2017. Joint deep modeling of users and items using reviews for recommendation. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, WSDM '17, 425–434. New York, NY, USA: ACM.