Feat2Vec: Supervised and Self-Supervised Embeddings of Data

Anonymous Author(s)
Affiliation
Address
email

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 for supervised learning where explicit labels are available, as well as for 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 significantly outperforms existing algorithms that do not leverage the structure of the 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.

1 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, and can be used to replace the original vector in downstream prediction tasks. Embeddings have multiple advantages, as they enable more efficient training **?**, and are often used for general-purpose reasoning..

We consider two kind of algorithms that use embeddings: **supervised methods**, like matrix factorization, produce embeddings that are highly tuned to a prediction task. These embeddings may be interpretable but do not usually generalize to other tasks. We refer to these embeddings as *task-specific*. So an embedding of a movie learned to predict ratings, it is unlikely to be extremely helpful for an unrelated prediction task. On the other hand, **self-supervised methods** (sometimes less precisely referred as unsupervised methods) like the Word2Vec models **?**, are designed to provide embeddings that are useful for a wide-array of downstream predictions tasks. For example, the embeddings may be used in analogy solving **?**, or sentiment analysis **?**.

In this paper we propose Feat2Vec as a novel method that allows calculating embeddings of arbitrary feature types 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. Second, we allow calculating embeddings of arbitrary features. Consider the work that a researcher needs to do to extend a Word2Vec model to support additional features. For example, to add document identifiers, the researchers that created the famous Doc2Vec **?** model, needed to design a new model and sampling strategy. In supervised tasks, this may be helpful to avoid the cold-start problem by representing an item with a description (e.g., text features).

Submitted to 32nd Conference on Neural Information Processing Systems (NIPS 2018). Do not distribute.

2 Preliminaries

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

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

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. The function ω is an activation. Choices for ω include a linear link ($\omega(x) = x$) for continuous outputs, or a logistic link ($\omega(x) = \frac{\exp(x)}{\exp(x)+1}$) for binary outputs.

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:

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

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. **?** 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 **?**.

3 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 (§ **??**).

3.1 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. 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:

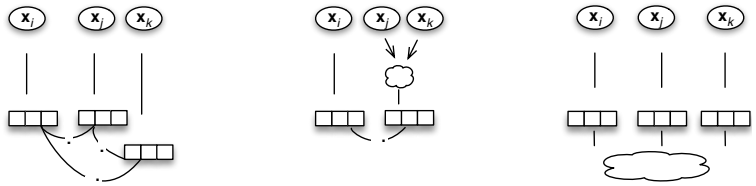
$$\hat{y}(\vec{x}, \vec{b}, \vec{\phi}) = \omega\left(b_0 + \sum_{i=1}^{n} b_i x_i\ +\ \sum_{i=1}^{|\vec{\kappa}|} \sum_{j=i}^{|\vec{\kappa}|} \phi_i(\vec{x}_{\vec{\kappa}_i}) \cdot \phi_j(\vec{x}_{\vec{\kappa}_j})\right) \tag{3}$$

In this notation, $\vec{x}_{\vec{\kappa}_i}$ is a subvector that contains all of the features that belong to the group $\vec{\kappa}_i$. Thus, $x_{\vec{\kappa}_i} = [x_j : j \in \vec{\kappa}_i]$. The intuition is that by grouping (sub-)features as a single entity, we can can reason on a higher level of abstraction. Instead of individual sub-features interacting among each other, the embeddings of feature groups interact with those of other groups. 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. Across groups, entities interact with each other via the output of ϕ only.

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 **??** compares existing factorization methods with our novel model. In this example, $\mathrm{Feat2Vec}$ 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.



(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.

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 shallow Factorization Machine by allowing each feature group to be a singleton : $\vec{\kappa} = \{\{x_1\}, \{x_2\} \ldots \{x_n\}\}$ and the linear extraction function presented in Equation **??**.

We can use $\mathrm{Feat2Vec}$ to both use large feature sets and 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, the group of word features. 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 **??** shows an approach of using neural networks within factorization machines that has been proposed multiple times **??**. 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. However, it would be straightforward to combine this approach with $\mathrm{Feat2Vec}$. This is not explored in this work.

3.2 Supervised Learning from Data

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

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

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; the hyperparameter γ controls the amount of regularization. For the labeling and classification tasks, we optimize the binary cross-entropy

It is straightforward to optimize Equation **??** directly. In the multi-class scenario, if the number of labels is very large, it is common practice use a binary classifier and use implicit sampling **?**. In this case we would have at least two feature groups—one of the feature groups is the label that we want to predict, and the other group(s) is the input from which we want to make the prediction. 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 **??**.

3.3 Self-supervised Learning From Data

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

The training dataset for a $\mathrm{Feat2Vec}$ model consists of only the observed data. In natural language, these would be documents written by humans. Since Feat2Vec (Equation **??**) 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. 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. In short, $\mathrm{Word2Vec}$ 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 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 samples. Unlike $\mathrm{Word2Vec}$, we do not constraint features types to be words. Features groups can be individual columns in a data matrix, but they need not to be. By grouping subfeatures using the parameter κ in Equation **??**, 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 following 2-step algorithm:

Algorithm 1 Implicit sampling algorithm for self-supervised $\mathrm{Feat2Vec}$: $\mathcal{Q}$

function FEAT2VEC_SAMPLE($S^+, k, \alpha_1, \alpha_2$)
  $S^- \leftarrow \emptyset$
  for $\vec{x}^+ \in S^+$ **do**
    Draw a random feature group $\kappa_i \sim \mathcal{Q}_1(\{\mathrm{params}(\phi_i)\}_{i=1}^{|\vec{\kappa}|}, \alpha_1)$
    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 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 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 need to 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:

$$P_{\mathcal{Q}_1}(\kappa_i|\,\mathrm{params}(\phi_i)\}_{i=1}^{|\vec{\kappa}|},\alpha_1) \;=\frac{\mathrm{params}(\phi_i)^{\alpha_1}}{\sum_{j=1}^{|\vec{\kappa}|}\mathrm{params}(\phi_j)^{\alpha_1}},\quad \alpha_1\in[0,1] \tag{6}$$

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 α_1 helps flatten the distribution. When $\alpha_1 = 0$, the feature groups are sampled uniformly, and when $\alpha_1 = 1$, they are sampled proportional to their complexity.

Sampling Feature Group Values. To sample a value from within a feature groups κ_i, we use a similar strategy to $\mathrm{Word2Vec}$ and use the empirical distribution of values:

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

Here, $\mathrm{count}(x)$ 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 possibilities: in the Negative Sampling that $\mathrm{Word2Vec}$ follows, the duplicate negative samples are simply ignored **?**. Alternatively, it is possible to account for the probability of random negative labels that are identical to positively labeled data using Noise Contrastive Estimation (NCE) **?**.

3.3.1 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 $\hat{y} = p(y = 1|\vec{x},\vec{\phi},\theta)$, expressed in Equation **??** to account for the possibility of random negative labels that appear identical to positively labeled data. θ here represents the parameters learned in during training (i.e. the b_i terms and parameters associated with the extraction functions ϕ_i in Equation **??**). Since we only deal with a dichotomous label, indicating a positive or negative sample, for self-superrvised learning, we restrict our attention to usage of Equation **??** 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. As a work-around, we appeal to the work of **?**, 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 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:

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

Table 1: Yelp rating prediction

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

where $s(.)$ denotes the score of a record $\vec{x}$ given parameter values/extraction functions:

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

and $P_{\mathcal{Q}}(.)$ 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:

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

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

$$L(S) = \arg\min_{\theta} \frac{1}{|S^+|} \sum_{\vec{x}^+ \in S^+} \Big(\log(\tilde{p}(y=1|\vec{x}^+, \vec{\phi}, \theta)) \; + \sum_{\vec{x}^- \sim \mathcal{Q}(\cdot|\vec{x}^+)}^{k} \log(\tilde{p}(y=0|\vec{x}^-, \vec{\phi}, \theta)) \Big) \tag{11}$$

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 features, one can use one of the feature as the target label, and the other 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, 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 group). We show the proof of this in the Supplemental Materials **??**.

4 Empirical Results

4.1 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 in **??**, 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 **??**. Instead of tuning the hyper-parameters, we follow previously published guidelines **?**.

4.1.1 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 **?**—which reportedly, is the state of the art for predicting customer ratings when textual reviews are available. To make results comparables, the Feat2Vec experiments

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

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 $\mathrm{Feat2Vec}$, we build 3 feature groups: item identifiers (in this case, restaurants), users and review text.

Table **??** compares our methods to DeepCoNN's published results because a public implementation is not available. We see that $\mathrm{Feat2Vec}$ 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 $\mathrm{Feat2Vec}$ 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.

4.2 General-purpose Embeddings

4.2.1 Does $\mathrm{Feat2Vec}$ 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 $\mathrm{Feat2Vec}$ algorithm and compare its performance in a targeted ranking task to $\mathrm{Word2Vec}$'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. In particular, in the movie dataset we compare the similarity of movie directors to those of actors who were cast in the same film for a left-out set of films. For our educational dataset, we compare rankings of textbooks by evaluating the similarity of textbook and 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.

4.2.2 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, 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.

4.2.3 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 most similar textbooks.

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

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

Table **??** 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%

4.3 Self-Supervised 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 expect this task to highlight Feat2Vec's advantage over token-based embedding learning algorithms, such as Word2Vec, since our rating embedding extraction function 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.

5 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 is general-purpose, and its embeddings are easily interpretable.

We recently discovered a promising direction for an recent method called StarSpace **?** with similar goals from ours. StarSpace does not allow self-supervised learning and is not tested with features beyond bag of words Nonetheless, a limitation of our work is that we do not compare with StarSpace, which future work may decide to do.

In the supervised setting, Feat2Vec is able to calculate embeddings for whole passages of texts, and we show experimental results outperforming an algorithm specifically designed for text—even when using the same feature extraction CNN. This suggests that the need for ad-hoc networks should be situated in relationship to the improvements over a general-purpose method.

In the self-supervised setting, Feat2Vec's embeddings are able to capture relationships across features that can be twice as better as Word2Vec's CBOW algorithm 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, 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]. Overall, we evaluate supervised and self-supervised Feat2Vec on 2 datasets each. Though further experimentation is necessary, we believe that our results are an encouraging step towards general-purpose embedding models.

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

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