Beyond Word Embeddings: Dense Representations for Multi-modal Data

Luis Armona[1,2], **José P. González-Brenes,**[2*] **Ralph Edezhath**[2*]
[1]Department of Economics, Stanford University,
[2]Chegg, Inc.
larmona@stanford.edu, {redezhath, jgonzalez}@chegg.com

Abstract

Methods that calculate dense vector representations for text have proven to be very successful for knowledge representation. We study how to estimate dense representations for multi-modal data (e.g., text, continuous, categorical). We propose Feat2Vec as a novel model that supports supervised learning when explicit labels are available, and self-supervised learning when there are no labels. Feat2Vec calculates embeddings for data with multiple feature types, enforcing that all embeddings exist in a common space. We believe that we are the first to propose a method for learning self-supervised embeddings that leverage the structure of multiple feature types. Our experiments suggest that Feat2Vec outperforms previously published methods, and that it may be useful for avoiding the cold-start problem.

1 Introduction

Informally, in machine learning a *dense representation*, or *embedding* of a vector $\vec{x} \in \mathbb{R}^n$ is another vector $\vec{\beta} \in \mathbb{R}^r$ 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 referred to as unsupervised methods) are not tuned for the prediction task they are ultimately used for. For example, word embedding algorithms such as Word2Vec (Mikolov et al. 2013b) are self-supervised. These algorithms are typically evaluated by analogy solving, or sentiment analysis (Le and Mikolov 2014), even though their loss functions are not tuned for either of these tasks.

Throughout this paper, we refer to any data that has different types of information as multi-modal. We propose Feat2Vec as a novel method that embeds multi-modal data, such as text, images, numerical or categorical data, in a common vector space—for both supervised and self-supervised scenarios. We believe that this is the first self-supervised algorithm that is able to calculate embeddings from data with multiple feature types. Consider the non-trivial work that was required 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 add a single feature (document ID). Feat2Vec is a general method that allows calculating embeddings for any number of features.

*Equal contribution

2 Feat2Vec

In this section we describe how Feat2Vec learns embeddings of feature groups.

2.1 Model

Feat2Vec predicts a target output $\tilde{y}$ from each observation $\vec{\mathbf{x}}$, which is multi-modal data that can be interpreted as a list of feature groups $\vec{g}_i$:

$$\vec{\mathbf{x}} = \langle \vec{g}_1, \vec{g}_2, \ldots, \vec{g}_n \rangle = \langle \vec{x}_{\vec{\kappa}_1}, \vec{x}_{\vec{\kappa}_2}, \ldots, \vec{x}_{\vec{\kappa}_n} \rangle \quad (1)$$

For notational convenience, we also refer to the i-th feature group as $\vec{x}_{\vec{\kappa}_i}$. In other words, each observation is constructed from a partition of raw features specified by $\vec{\kappa}$. Examples of how to group features include "text" from individual words, or an "image" from individual pixels. We define an entity to be a particular value or realization of a feature group.

We define our model for a target y as follows:

$$\tilde{y}(\vec{x}, \vec{\boldsymbol{\phi}}, \vec{\boldsymbol{\theta}}) = \omega\left(\sum_{i=1}^{n} \sum_{j=i}^{n} \overbrace{\phi_i(\vec{x}_{\vec{\kappa}_i}; \vec{\theta}_i)}^{\substack{\text{group } i \\ \text{embedding}}} \cdot \overbrace{\phi_j(\vec{x}_{\vec{\kappa}_j}; \vec{\theta}_j)}^{\substack{\text{group } j \\ \text{embedding}}} \right) \quad (2)$$

The number of dimensions d_i of each feature group may vary, but all of them are embedded to the same space via their *feature extraction function* $\phi_i : \mathbb{R}^{d_i} \to \mathbb{R}^r$. These functions learn how to transform each input (feature group) into an r-dimensional embedding of length r. We only allow a feature extraction function ϕ_i to act on the features of g_i; and feature groups interact with each other only via the output of ϕ_i. We denote the parameters of the extraction function ϕ_i as $\vec{\theta}_i$. ω is an activation (link) function. Intuitively, the dot product ($\cdot$) returns a scalar that measures the (dis)similarity between the embeddings of the feature groups.

A simple implementation of ϕ_i is a linear fully-connected layer, where the output of the r-th entry is:

$$\phi_i\left(\vec{x}_i; \vec{\theta}_i\right)_r = \sum_{a=1}^{d_i} \theta_{i_{r_a}} x_{i_a} \quad (3)$$

where $\vec{\theta}_{i,r}$ are learned weights. Other functional forms of ϕ_i we have explored in our experiments include:

- A convolutional neural network that transforms a text sequence into an r-dimensional embedding. This function is able to preserve information in the order of a text sequence, unlike other methods for transforming text to numerical values such as bag-of-words.
- A deep, fully connected neural network that projects a scalar, such as average rating, into an r-dimensional embedding space. This extraction function, by treating the input as numerical instead of categorical, requires inputs close in value (e.g. 4.01 and 4.02 star rating) to have similar embeddings.

Any neural function that outputs an r-dimensional vector could be plugged into our framework as an extraction function ϕ_i. Although in this work we have only used datasets with natural language, categorical and numerical features, it would be straightforward to incorporate other types of data–e.g. images or audio–with the appropriate feature extraction functions, enabling the embedding of multi-modal features in the same space.

Figure 1 compares existing factorization methods with our novel model. Figure 1a shows a Factorization Machine (Rendle 2010) that embeds each feature. Figure 1b shows Feat2Vec with two feature groups: the first group only has a single feature which is projected to an embedding (just like matrix factorization); but the second group has multiple features, which are together projected to a single embedding. 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. The caveat of this architecture is that is no longer possible to interpret the embeddings as latent factors related to the target task.

Our work in Feat2Vec significantly expands previously published models. For example, Principal Component Analysis (PCA) is a common method to learn embeddings of individual dimensions of data. Although structured regularization techniques exist (Jenatton, Obozinski, and Bach 2010), it is not obvious how to combine different dimensions in PCA when we are interested in treating a subset of dimensions as a single group. In traditional factorization methods, item identifiers are embedded and thus need to be observed during training (i.e., cold-start problem). Feat2Vec can learn an embedding from an alternative characterization of an item, such as a textual description. Feat2Vec extends Factorization Machine (Rendle 2010), in that we allow calculating embeddings for feature groups and we use feature extraction functions. StarSpace (Wu et al. 2018) introduces feature groups (called entities in their paper), but constrains them to be "bags of features," is supervised, and limited to two feature types. In contrast, Feat2Vec allows continuous features, and does not require the user to assign one feature as a specific target. Additionally, we propose a self-supervised learning algorithm.

2.2 Supervised Learning from Data

We can learn the the parameters of a Feat2Vec model $\vec{\theta}$ by minimizing a supervised loss function $\mathcal{L}_{\text{sup}}$:

$$\arg\min_{\vec{\theta}} \sum_{\vec{x}\in\mathbf{X}} \mathcal{L}_{\text{sup}}\big(\, y(\vec{x}), \tilde{y}(\vec{x}; \mathbf{\Phi}, \vec{\theta})\,\big) \tag{4}$$

Here, $\mathbf{X}$ is the training dataset, $y(\vec{x})$ is the true target value observed in the data for an observation $\vec{x} \in \mathbf{X}$, and $\tilde{y}(\vec{x})$ is its estimated value (using Equation 2); $\vec{\theta}$ represents the parameters learned during training (i.e. the parameters associated with the extraction functions ϕ_i). For labeling and classification tasks, we optimize the logistic loss.

We can optimize Equation 4 directly using gradient descent algorithms. Multi-class classification models can be computationally costly to learn when the number of classes is very high. In these cases, a common work-around is to reformulate the learning problem into a binary classifier with implicit sampling (Dyer 2014): simply redefine the observation by appending the target label as one of the feature groups of the observation (e.g., rating), while keeping the other feature group(s) the same (e.g. review text). The new implicit target label is a binary value that indicates whether the observation exists in the training dataset.

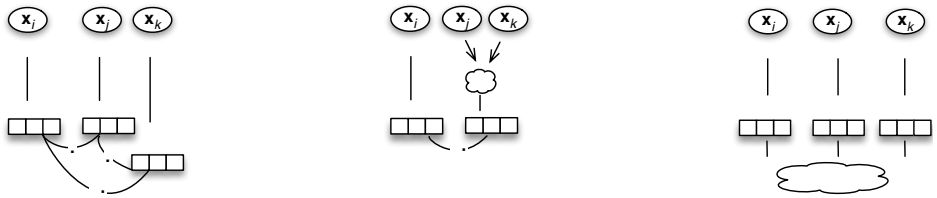
In implicit sampling, instead of using all of the possible negative labels, one samples a fixed number (k) from the set of possible negative labels for each positively labelled record. Implicit sampling is also useful for multi-label classification with a large number of labels, or when negative examples are not easily available. We leverage on implicit sampling for our self-supervised learning algorithm.

2.3 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 the observed data. In natural language, these would be documents written by humans, along with document metadata. Since self-supervised Feat2Vec will require positive and negative examples during training, we supply unobserved data as negative examples. This sampling procedure is analogous to the Word2Vec algorithm, which 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. Unlike Word2Vec, we do not constrain features types to be words. Instead, by leveraging the feature groups of the data defined by $\vec{\kappa}$ (Equation 2), the model can reason on more abstract entities in the data. For example, in our experiments on a movie dataset, we define a "genre" feature group, where we group non-mutually exclusive indicators for movie genres, including comedy, action, and drama films.

We start with a training dataset $\mathbf{X}$ of records with n feature groups. We assign a positive label to each observation in $\mathbf{X}$. For each observed (positive) datapoint $\vec{x}^{+}$, we generate k observations with negative labels using the 2-step algorithm documented in Algorithm 1. We illustrate the intuition of this algorithm with an example. Consider a dataset



(a) **Factorization Machine** (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.

with three feature groups—entities for (i) a passage of text (bag of words), (ii) a restaurant identifier (categorical), and a sentiment (a continuous value). The positive examples are observations seen in the training set. Negative examples are generated by modifying positive examples, specifically by substituting one of the entities randomly from another observation. For example, by substituting the passage of text, a negative example for the observation $\langle$*Food was great.*, *Le Fancy Restaurant*, $100\rangle$ may result as $\langle$*Ewww, gross!!!*, *Le Fancy Restaurant*, $100\rangle$.

Algorithm 1 Implicit sampling algorithm for self-supervised Feat2Vec

function $\mathcal{Q}(\vec{x}^+; k, \alpha_1, \alpha_2)$
  $\mathbf{X}^- \leftarrow \emptyset$
  for $j \in \{1, \ldots, k\}$ **do**
    Draw which feature group $i \sim \mathcal{Q}_1(\mathbf{\Phi}, \alpha_1)$ to sample
    Draw a random entity $\vec{g} \sim \mathcal{Q}_2(\mathrm{X}_{\kappa_i}, \alpha_2)$
    $\vec{x}^- \leftarrow \vec{x}^+$ ▷ Set identical to the positive sample
    $\vec{x}^-_{\kappa_i} \leftarrow \vec{g}$ ▷ substitute the i-th feature group with $\vec{g}$
    $\mathbf{X}^- \leftarrow \mathbf{X}^- + \{\vec{x}^-\}$ ▷ append
  end for
  return $\mathbf{X}^-$ ▷ Sampled negative observations from $\vec{x}^+$
end function

More formally, our negative sampling algorithm generates an observation with a negative label from an observation in the training dataset as follows. First, it randomly selects the i-th feature group from a noise distribution $\mathcal{Q}_1(\cdot)$. It then creates a negative observation that is identical to $\vec{x}^+$, except that its i-th feature group value 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 noise distributions that we use.

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 i from a multinomial distribution with probabilities proportional to a feature group's complexity. By complexity, we mean the number of parameters we learn that are associated with a feature group's extraction function ϕ_i. This sampling procedure 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:

$$P_{\mathcal{Q}_1}(i|\mathbf{\Phi}, \alpha_1) = \frac{\mathrm{params}(\phi_i)^{\alpha_1}}{\sum_{j=1}^{|\vec{\kappa}|} \mathrm{params}(\phi_j)^{\alpha_1}} \quad (5)$$

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 $\alpha_1 \in [0, 1]$ flattens 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 i, we use a similar strategy to Word2Vec and use the empirical distribution of values:

$$P_{\mathcal{Q}_2}(\vec{g}|\mathrm{X}_{\kappa_i}, \alpha_2) = \frac{\mathrm{count}(\vec{g})^{\alpha_2}}{\sum_{\vec{j} \in \mathrm{X}_{\kappa_i}} \mathrm{count}(\vec{j})^{\alpha_2}}, \quad \alpha_2 \in [0, 1] \quad (6)$$

Here, $\mathrm{count}(\vec{g})$ is the number of times an entity $\vec{g}$ of feature group i appeared in the training dataset $\mathbf{X}$. Thus, an entity is sampled according to its empirical distribution—whether it is a single word, a continuous number, or a passage of text. Again, α_2 is simply 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. We need to make sure that our loss function is well-behaved distribution that integrates to 1. This often requires a partition function $Z_{\vec{x}}$ for each unique record type $\vec{x}$, which can be computationally costly and greatly increase the complexity of our model. Instead, we appeal to the work of Mnih and Teh (2012), who show that in models with many parameters setting $Z_{\vec{x}} = 1$ in advance does not change the performance of the model. The intuition is that if the underlying model has enough free parameters, it

will effectively learn the probabilities itself, since systemic under/over prediction of probabilities will result in penalties on the loss function.

For self-supervised learning, we adjust the structural statistical model of Equation 2 to account for the probability of negative examples, as in (Dyer 2014). Written explicitly, this is:

$$\tilde{p}(y=1|\vec{x},\vec{\boldsymbol{\phi}},\vec{\theta}) = \frac{\exp\big(\tilde{y}(\vec{x}|\vec{\boldsymbol{\phi}},\vec{\theta})\big)}{\exp\big(\tilde{y}(\vec{x},\vec{\boldsymbol{\phi}},\vec{\theta})\big) + k \times P_{\mathcal{Q}}(\vec{x}|\alpha_1,\alpha_2)} \triangleq \hat{s}(\vec{x};\vec{\theta}) \quad (7)$$

For notational convenience, we refer to this structural probability as $\hat{s}(\vec{x};\vec{\theta})$. Here, $\tilde{y}(.)$ denotes the model in Equation 2 with ω set to the identity function. This can be interpeted as the "score" of an observation with respect to Feat2Vec. $P_{\mathcal{Q}}(x)$ denotes the probability of x under noise distribution $\mathcal{Q}$; it accounts for the possibility of sampling an observation with a negative label from Algorithm 1 that appears identical to one in the training data (with a positive label). The probability of a negatively labeled record $\vec{x}$, conditional on a positive datapoint $\vec{x}^+$, is simply given by:

$$P_{\mathcal{Q}}(\vec{x}|\alpha_1,\alpha_2,\mathbf{X},\vec{x}^+) = P_{\mathcal{Q}_2}(\vec{x}_{\kappa_i}|\mathrm{X}_{\kappa_i},\alpha_2)\times P_{\mathcal{Q}_1}(i|\,\mathrm{params}(\phi_i)\}_{i=1}^n,\alpha_1) \quad (8)$$

We seek to optimize the parameters $\vec{\theta}$ of the feature extraction functions $\vec{\boldsymbol{\phi}}$:

$$\arg\min_{\vec{\theta}} \sum_{\vec{x}^+\in\mathbf{X}} \mathcal{L}_{\text{self}}\big(\vec{x}^+;\vec{\theta}\big) \quad (9)$$

For this, we define our self-supervised loss $\mathcal{L}_{\text{self}}$ as:

$$-\sum_{\vec{x}^+\in\mathbf{X}}\left(\log\big(\hat{s}(\vec{x}^+;\vec{\theta})\big) + \sum_{\vec{x}^-\sim\mathcal{Q}(\cdot|\vec{x}^+)}^{k}\log\big(1-\hat{s}(\vec{x}^-;\vec{\theta})\big)\right) \quad (10)$$

Note that the observations with positive labels come directly from the training data, and the observations with negative labels are sampled using Algorithm 1. With n feature groups, the loss function of self-supervised Feat2Vec can be shown to be a weighted average of the losses from n distinct supervised Feat2Vec models,[1] one for each feature group. In other words, it optimizes n multi-label classifiers, where each classifier is optimized for a different feature group. The intuition is as follows. In Algorithm 1, we first sample a target feature group i according to $P_{\mathcal{Q}_1}$, and then add the loss from a supervised Feat2Vec model for i to the total self-supervised loss. Thus, $P_{\mathcal{Q}_1}$ determines the weights the self-supervised loss function assigns to each feature group.

3 Empirical Results

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.

[1]The proof can be found in supplementary material at http://web.stanford.edu/~larmona/feat2vec/supplemental_theorem.pdf

3.1 Supervised Embeddings

To evaluate our supervised learning model, we compare against two baselines: (i) a method called DeepCoNN (Zheng, Noroozi, and Yu 2017),which is a deep network specifically designed for incorporating text into recommendation problems—reportedly, it is the state of the art for predicting customer ratings when textual reviews are available; (ii) and with Matrix Factorization (Koren, Bell, and Volinsky 2009), a commonly used baseline.

We use our own implementation of Matrix Factorization, but we rely on DeepCoNN's published results (at time of writing, their implementation was not available). Fortunately, DeepCoNN also compares on Matrix Factorization, which makes interpretation of our results easier. To make results comparable, the Feat2Vec experiments use the same feature extraction reported by DeepCoNN (a convolutional neural network architecture). Thus, we can rule out that differences of performance are due to choices in feature extraction functions. Instead of tuning the hyper-parameters, we follow previously published guidelines (Zhang and Wallace 2017).

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. For Feat2Vec, we build 3 feature groups: item (restaurant) identifiers, user identifiers, and review text.

Table 1 summarizes our results. We use mean squared error (MSE) as the evaluation metric. Feat2Vec provides a large performance increase. Additionally, we claim that our approach is more general, because it can trivially be extended to incorporate more features. It is also more efficient: DeepCoNN relies on concatenating text, and when the average reviews per user is $\bar{n_u}$ and the average reviews per item is $\bar{n_i}$, this results in text 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 Feat2Vec, can be 1-2 orders of magnitude more efficient for datasets where $\bar{n_i} \times \bar{n_u}$ is large.

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	-

3.2 Self-Supervised Embeddings

The prediction of similar or related words is a commonly used method for evaluating self-supervised word embeddings (Bojanowski et al. 2017; Mikolov et al. 2013a). We generalize this task for feature types beyond text as follows.

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

Given a dataset with n feature groups we have trained a self-supervised Feat2Vec model on, we choose a single group i as the "target" and a group j as the predictor. Given a held-out instance $\vec{x}$, we rank all possible values of the target group in X_{κ_i} by cosine similarity to the source group's embedding $\phi_j(\vec{x}_{\vec{\kappa}_j})$ for that instance. We then retrieve the ranking of the actual target entity in the test dataset relative to cosine similarity scores for all other possible entities in $\vec{\kappa}_i$. Rankings are evaluated according to their mean percentile rank (MPR): $MPR = \frac{1}{N}\sum_{i=1}^{N} R_i/(\max_i R_i)$, where R_i is the rank of the entity for observation i under our evaluation procedure, and $\max_i R_i$ is the worst ranking that can be received. A score of 0 would indicate perfect performance (i.e. top rank every test sample given), so lower is better under this metric.

Under this evaluation, we will compare the performance of Feat2Vec algorithm to Word2Vec's CBOW algorithm for learning embeddings. During training, CBOW uses a "context window" of nearby words to predict a target word, so there is a natural way to apply embeddings learned from CBOW to our ranking task (aggregate context embeddings in the left-out instance's source group to predict the target group). To create a corpus for training Word2Vec, each record in the data is treated as a separate document. Specifically, we tokenize all feature values in a record and treat them as words (e.g. a movie record flagged as belonging to the horror genre has the token `genre_horror` in its document). We set the Word2Vec context window wide enough so that training is invariant to the token order. We use hyperparameters $\alpha_2 = .75$ and $k = 5$ from (Mikolov et al. 2013b) and set the embeddings dimension to $r = 50$. These hyperparameters are used in each of our embedding models during evaluation. The novel feature group weighting parameter in Feat2Vec was set to $\alpha_1 = 0.25$ in order to highlight the differences with Word2Vec, since $\alpha_1 = 1$ corresponds to all feature groups being sampled equally. This parameter α_1 was not tuned further.

We evaluate self-supervised Feat2Vec on 3 datasets:

- **Movies** The Internet Movie Database (IMDB) is a publicly available dataset[3] of information related to films, television programs and video games. We limit our experiments to data on its 465,136 movies. Table 2 summarizes our feature groups. For groups with multiple categories, we use a "Bag of categories" approach. For example, we sum embeddings of individual actors such as Tom Cruise to produce a single "actor" feature group embedding associated with a movie.
- **Education** We use a dataset from a 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.
- **Yelp** We use the Yelp dataset from our supervised experiments to evaluate the efficacy of self-supervised embeddings in ratings prediction. We train embeddings for the following feature groups: user ID, restaurant ID, review text, number of times a review is flagged as funny, and 0-5 star rating of the review.

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

Table 2: IMDB Feature Groups

Feature Type Name	Type	# of features
Runtime (minutes)	Real-valued	1
IMDB rating (0-10)	Real-valued	1
# of rating votes	Real-valued	1
Is adult film?	Boolean	2
Movie release year	Categorical	271
Movie title	Text	165,471
Directors	Bag of categories	174,382
Genres	Bag of categories	28
Writers	Bag of categories	244,241
Principal actors	Bag of categories	1,104,280

Results After training IMDB embeddings, we use cast members in a held out set of movies to predict the actual director of the film. We rank directors by cosine similarity of their embedding to the title's cast member feature group embedding. For the educational dataset, we directly retrieve the most similar textbooks to the user embedding. Table 3 presents our evaluation results. Feat2Vec outperforms CBOW in the MPR metric. Additionally, while CBOW predicts the actual director 1.26% of the times, Feat2Vec does so 2.43% of the time, a 92% improvement in Top-1 Precision over CBOW.

Table 3: 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 our estimated self-supervised Feat2Vec model performs when predicting a real-valued feature. We expect this task to highlight Feat2Vec's advantage over token-based embedding learning algorithms, such as Word2Vec. For IMDB ratings, we use a 3-layer DNN extraction function that will require embeddings of numerically similar ratings to be close, while Word2Vec will treat two numerically different ratings as completely different entities. We evaluate the prediction of IMDB ratings by choosing the rating embedding most similar[4] to the embedding of the movie's director, and compute the MSE of the predicted rating in the test dataset. Feat2Vec scores an MSE of 6.6, while Word2Vec (CBOW) scores 9.3.

We also use the Yelp dataset, predicting the most similar rating embedding to the review text embeddings produced by Feat2Vec, Word2Vec (CBOW), and Doc2Vec (DM). Doc2Vec is only used for the Yelp dataset because there is no obvious "base document" in the IMDB dataset for Doc2Vec to learn on, while in the Yelp dataset the review text is a natural candidate. For Word2Vec and Doc2Vec, the review text

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

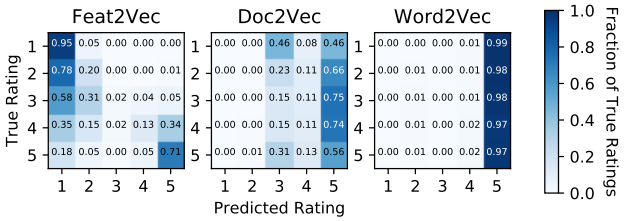


Figure 2: Confusion Matrices of Self-Supervised Yelp Ratings Predictions

embedding is the average of word embeddings, analogous to the context vector used for learning in these algorithms. Figure 2 reports the confusion matrices for each model from this experiment. Word2Vec is poor in its predictive power, predicting 5 stars for 97% of the test sample. Though Feat2Vec and Doc2Vec yield comparable MSE (2.94 vs. 2.92, respectively), Feat2Vec outperforms Doc2Vec by a substantial margin in classification error rate (55% vs. 73%) and mean absolute error (1.13 vs. 1.31). In general, Doc2Vec is unable to identify low rating reviews: only 0.4% of Doc2vec predictions are ≤ 2 stars, despite this comprising 20% of the data. In contrast, Feat2Vec is more diverse in its predictions, and better able to identify extreme reviews.

4 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 embedding methods. 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 outperforms an algorithm specifically designed for text—even when using the same feature extraction function. In the self-supervised setting, Feat2Vec exploits the structure of a dataset to learn embeddings in a more sensible way than existing methods. This yields performance improvements in our ranking and prediction tasks. To the extent of our knowledge, self-supervised Feat2Vec is the first method to calculate continuous representations of data with multi-modal feature types and no explicit labels.

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 forward towards general-purpose embedding models.

[5]The Feat2Vec model code is available here: `https://github.com/CheggEng/Feat2Vec`. The experiments using non-proprietary data are available here: `http://web.stanford.edu/~larmona/feat2vec/`.

References

Bojanowski, P.; Grave, E.; Joulin, A.; and Mikolov, T. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics* 5:135–146.

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. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 1725–1731.

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.

Jenatton, R.; Obozinski, G.; and Bach, F. 2010. Structured sparse principal component analysis. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 366–373.

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.; Chen, K.; Corrado, G.; and Dean, J. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013b. 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 *Proceedings of the International Conference on Machine Learning*.

Rendle, S. 2010. Factorization machines. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, 995–1000. Washington, DC, USA: IEEE Computer Society.

Wu, L. Y.; Fisch, A.; Chopra, S.; Adams, K.; Bordes, A.; and Weston, J. 2018. Starspace: Embed all the things! In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Zhang, Y., and Wallace, B. 2017. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 253–263. Asian Federation of Natural Language Processing.

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.