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 text have proven to be very successful for knowledge representation. We study how to estimate dense representations when multiple feature types exist (e.g., text, continuous, categorical) within a dataset. 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 different feature types 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 that they are ultimately used for. For example, Word2Vec (Mikolov et al., 2013) methods are self-supervised and are often used for transfer learning. Self-supervised embeddings are evaluated in how useful they are for analogy solving, or sentiment analysis (Le and Mikolov, 2014) eventhough the loss function is not tuned for either of these tasks.

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. We believe that this is the first general-purpose 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.

2 Feat2Vec

Principal Component Analysis 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 when a group of dimensions need to be treated as a single group. For example, consider a dataset that has a textual description of items— we may be interested in having an embedding for the feature group (the description) instead of the individual words. In this section we describe how Feat2Vec learns embeddings of feature groups.

2.1 Model

Feat2Vec predicts a target output $\hat{y}$ from a list of feature groups constructed from a partition $\vec{\boldsymbol{\kappa}}$ of raw features $\vec{\mathbf{x}}$:

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

Each of the n feature groups $\vec{g}_i$ is defined a priori by $\vec{\kappa}_i$, which indexes the dimensions in $\vec{\mathbf{x}}$ that belong to the i-th group. The number of dimensions of each group may vary, but all of the feature groups are embedded to the same space via their *feature extraction function* ϕ_i. These functions learn how to embed each vector (feature group) and outputs a vector in $\mathbb{R}^r$. More, formally:

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

Here ω is an activation 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:

$$\phi_{i,r}\left(\vec{x_i};\vec{\theta}_r\right)_r = \sum_{a=1}^{d_i} \theta_{r_a} x_{i_a} \tag{3}$$

In traditional factorization methods, like Matrix Factorization, each item identifier is embedded. Thus, all of the items need to be observed during training (i.e., cold-start problem). Feat2Vec could be used to predict on unseen items, by learning an embedding from an alternative characterization of the item—for example an item's textual description. For this, we can treat the entire textual description as a feature group κ_i. A feature extraction function ϕ acts on the features in κ_i, and the other features interact with the words only via the output of ϕ. However, Feat2Vec is not limited to bag of words features, and we demonstrate this with continuous features.

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. 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. The caveat of this architecture is that is no longer possible to interpret the embeddings as latent factors related to the target task.

The structure of Feat2Vec model significantly expands previously published models. For example, it improves Factorization Machine (Rendle, 2010), in that we allow calculating embeddings for feature groups and feature extraction functions. StarSpace (Wu et al., 2017) introduces feature groups (they call entities), but are constrained to be a "bag of features". Feat2Vec allows continuous features. StarSpace is limited to only two feature types– where one feature is treated as a label, and the other is a bag of features, and there is no know self-supervised training method. In contrast, our model does not require the user to assign a feature as a label, and we propose a self-supervised learning algorithm.

2.2 Supervised Learning from Data

We can learn the the parameters of a Feat2Vec model $\boldsymbol{\theta}$ using training data by minimizing a loss function $\mathcal{L}$:

$$\arg\min_{\vec{\theta}} \sum_x \mathcal{L}\big(y(x), \hat{y}(x;\vec{\theta})\big) + \gamma||\boldsymbol{\theta}||^w \tag{4}$$

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 (Equation 2); $\boldsymbol{\theta}$ represents the parameters learned in during training (i.e. the parameters associated with the extraction functions ϕ_i (for example 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 4 directly. In the multi-class scenario, if the number of labels is very large, it is common practice to 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 in the data 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 k negative labels equal for each positive label. 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 and Freudenthaler, 2014).

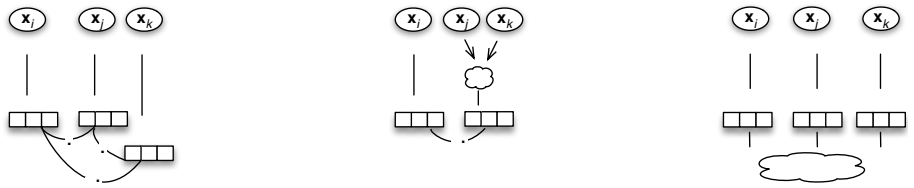
2.3 Self-supervised Learning From Data

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 Feat2Vec (Equation 2) 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. Instead, by grouping subfeatures using the parameter $\vec{\boldsymbol{\kappa}}$ in Equation 2, 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 define a "genre" feature group, where we group non-mutually exclusive indicators for movie genres, including comedy, action, and drama films.



(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.

We start with a dataset S^+ of records with n feature groups. We then mark all observed records in the training set as our positive-labeled 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 $j \in \{1, \dots, k\}$ **do**
      $\vec{x}^- \leftarrow \vec{x}^+$ ▷ set initially to be equal to the positive sample
      Draw a random entity $\tilde{x} \sim \mathcal{Q}_2(\mathrm{X}_{\kappa_i}, \alpha_2)$
      $\vec{x}^-_{\kappa_i} \leftarrow \tilde{x}$ ▷ substitute the i-th feature group 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 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. Let $P_{\mathcal{Q}}(x)$ denote the probability of x under 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 feature group's extraction function ϕ_i. 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:

$$P_{\mathcal{Q}_1}(\kappa_i | \text{params}(\phi_i)\}_{i=1}^{|\vec{\kappa}|}, \alpha_1) = \frac{\text{params}(\phi_i)^{\alpha_1}}{\sum_{j=1}^{|\vec{\kappa}|} \text{params}(\phi_j)^{\alpha_1}} \tag{5}$$

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 κ_i, we use a similar strategy to Word2Vec and use the empirical distribution of values:

$$P_{\mathcal{Q}_2}(x | \mathrm{X}_{\kappa_i}, \alpha_2) = \frac{\text{count}(x)^{\alpha_2}}{\sum_{x'_{\kappa_i} \in S^+} \text{count}(x'_{\kappa_i})^{\alpha_2}}, \quad \alpha_2 \in [0, 1] \tag{6}$$

Here, $\text{count}(x)$ 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 2, 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 2 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 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.

Written explicitly, the new structural probability model is:

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

$$s(\vec{x}, \vec{\phi}, \boldsymbol{\theta}) = \sum_{i=1}^{|\vec{\kappa}|} \sum_{j=i}^{|\vec{\kappa}|} \phi_i(\vec{x}_{\vec{\kappa}_i}) \cdot \phi_j(\vec{x}_{\vec{\kappa}_j}) \quad (8)$$

$s(.)$ 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:

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

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. With n feature groups, self-supervised Feat2Vec can be shown to be equivalent to optimizing a convex combination of the loss functions of n supervised Feat2Vec models with implicit sampling (proof omitted). In other words, it optimizes n multi-label classifiers, where each classifier is optimized for a different target (i.e. feature group). Intuitively, during sampling in self-supervised Feat2Vec, we choose a target feature group according to $P_{\mathcal{Q}_1}$, and then add that group's supervised loss to the total loss.

3 Empirical Results

3.1 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, 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.

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	-

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 (restaurant) and user identifiers, and review text.

Table 1 compares our methods to DeepCoNN's published results because a public implementation is not available. Feat2Vec provides a large performance increase when comparing the reported improvement in the mean squared error (MSE) over Matrix Factorization, 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 $\bar{n_i} \times \bar{n_u}$

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

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

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.

3.2 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). $MPR = \frac{1}{N}\sum_{i=1}^{N} R_i/(\max R)$, 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. 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 film 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.

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 of the film. 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 directly retrieve the most similar textbooks to the user embedding.

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%

3.3 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 (scale is 0 to 10) in the test dataset by choosing the IMDB 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 scores 9.3.

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. Figure 2 reports the confusion matrices for each model from this experiment. In general, 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). As made evident by the figure, 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

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

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

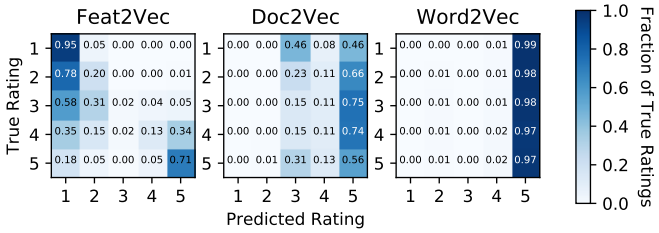


Figure 2: Confusion Matrices of Self-Supervised Yelp Ratings Predictions

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. To the extent of our knowledge, Self-Supervised Feat2Vec is the first method able to calculate continuous representations of data with arbitrary feature types without 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 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*.

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.

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

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.

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.

Wu, L.; Fisch, A.; Chopra, S.; Adams, K.; Bordes, A.; and Weston, J. 2017. Starspace: Embed all the things! *arXiv preprint arXiv:1709.03856*.

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.