FEAT2VEC: SEMANTIC MODULARITY FOR CONTINUOUS REPRESENTATIONS OF DATA WITH FEATURES

Anonymous authors
Paper under double-blind review

ABSTRACT

Methods that calculate dense embeddings for features in unstructured data—such as words in a document—have proven to be very successful for knowledge representation. Surprisingly, very little work has focused on methods for structured datasets where there is more than one feature type—this is, datasets that have arbitrary features beyond words.

We study how to estimate continuous representations for multiple feature types within a dataset, where each feature type exists in a different higher-dimensional space. Feat2Vec is a novel method that calculates embeddings for data with multiple feature types enforcing *semantic modularity* across features—all different feature types exist in a common space. We demonstrate our work on two datasets. The first one is collected from a leading educational technology firm, and thus we are able to discover a common continuous space for entities such as universities, courses, textbooks and students. The second dataset is the public IMDB dataset, and we discover embeddings for entities that include actors, movies and producers. Our experiments suggest that Feat2Vec significantly outperforms existing algorithms that do not leverage the structure of the data.

1 PRELIMINARIES

Informally, in machine learning an *embedding* of a vector $\vec{x} \in \mathbb{R}^n$ is another vector $\vec{y} \in \mathbb{R}^r$ that has much lower dimensionality ($r \ll n$) than the original representation, and can be used to replace the original vector in downstream prediction tasks. Embeddings have multiple advantages, as they enable more efficient training (Mikolov et al., 2013b), and unsupervised learning (Schnabel et al., 2015). For example, when applied to text, semantically similar words are mapped to nearby points.

Word2Vec (Mikolov et al., 2013a;b) is an extremely successful software package that contains two embedding functions with the same domain and codomain:

$$\text{Word2Vec} : \left\{ \vec{x} \in \mathbb{R}^n \mapsto \vec{y} \in \mathbb{R}^r \right\} \tag{1}$$

Word2Vec is suited for calculating embeddings for datasets that consist of documents of words with a vocabulary size of n. Here, $\vec{x}$ is sparse because words (and categorical variables, in general) are modeled using one-hot encoding. In this paper we study how to generalize embedding functions for arbitrary datasets with multiple feature types by leveraging their internal structure. We call this approach Feat2Vec:

$$\text{Feat2Vec} : \left\{ \vec{x} \in \vec{\mathcal{F}} \mapsto \vec{y} \in \mathbb{R}^r \right\} \tag{2}$$

$$\vec{\mathcal{F}} = \left[\mathbb{R}^{d_1}, \mathbb{R}^{d_2}, \ldots, \mathbb{R}^{d_n} \right] \tag{3}$$

Here, $\vec{\mathcal{F}}$ is a structured feature set—where each dimension represents a different feature type. Notice that is possible to use Word2Vec on structured data by simply flattening the input. However, Feat2Vec leverage the features' structure to enforce *semantic modularity* across features. That is, the embedding of an observation is additive in respect to the embeddings of its feature types; thus, each feature type $\mathcal{F}_i$ must be projected into the same embedding space. Semantic modularity is useful because it enables making principled comparisons between different feature types. For example, on an educational dataset that contains many feature types, including students and courses, semantic modularity enables inferring course preferences for students.

2 FEAT2VEC

The rest of this section describes the three parts of a Feat2Vec implementation. § 2.1 describes a model that infers an embedding from an observation. To learn this model we need positive and negative examples: a positive example is "semantic" and it is observed during training, but negative examples are not. This is similar how a positive example in Word2Vec are grammatical co-ocurrences, and negative examples are generated. § 2.2 describes our novel sampling strategy. Finally, in § 2.3 we describe how to learn a Feat2Vec model from data.

2.1 STRUCTURED DEEP-IN FACTORIZATION MACHINE

Levy & Goldberg (2014) showed that a Word2Vec model can be formalized as a Shallow Factorization Machine (Rendle, 2010) with two features types: a word and its context. This factorization model is a binary classifier that scores the likelihood of an observation $\vec{x} \in \mathbb{R}^n$ being labeled $y \in \{0, 1\}$, as proportional to the sum of the factorized pairwise interactions:

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

Here, Y is the random variable that defines whether the observation is "semantic" (which in practice means whether it occurs in the training data), $\vec{\beta}_i$ is a rank-r vector of factors, and σ is a sigmoid:

$$\sigma(x) = \log\left(\frac{\exp(x)}{\exp(x+1)}\right) \tag{5}$$

However, features may have some structure that is known beforehand. For example, consider multiple discrete entity types, like universities, students, and books. To model this data, a Shallow Factorization Machine would ignore the structure and simply concatenate the one-hot encoding representation of each entity into a single feature vector. Feat2Vec relies on an novel extension to factorization called Structured Deep-In Factorization Machine, which we describe in detail in a companion paper (Anonymized, Under review). While Shallow Factorization Machine learns an embedding per feature, the Structured Deep-In model allows greater flexibility. For example, consider using images or text on these factorization models. The shallow model learns an embedding for each word, or an embedding per pixel. The structured model enables higher-level of abstraction and flexibility, and it can learn an embedding per passage of text, or an embedding per image.

Structured Deep-In Factorization Machine inputs $\vec{\kappa}$ as a vector of vectors that define the structure of the groups of features. Each entry i is a vector of size d_i and it is used to represent the feature types of $\mathcal{F}_i$. In a shallow model, a feature interacts with all other features, but in the structured model they only interact with features from different groups. For example, consider a model with four features. If we define $\vec{\kappa} = \big[\,[1, 2], [3, 4]\,\big]$, the structure of the model would not allow x_1 to interact with x_2, or x_3 to interact with x_4. Additionally, for each group of features, the model allows applying a $d_i \times r$ feature extraction function ϕ_i. More formally, this is:

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

In this notation, $\vec{x}_{\vec{\kappa}_i}$ is a subvector that contains all of the features that belong to the group κ_i. Thus, $\vec{x}_{\vec{\kappa}_i} = \big[\, x_{i_a} | a \in \{1, 2, \ldots, |\kappa_i|\}\,\big]$. The simplest implementation for ϕ_i is a linear fully-connected layer, where the output of the r-th entry is:

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

We build the Feat2Vec embedding for an observation $\vec{x}$ as:

$$\text{Feat2Vec}(x) = \big[\ \phi_q(\vec{x}_{\vec{\kappa}_1}),\ \phi_1(\vec{x}_{\vec{\kappa}_2}),\ \ldots,\ \phi_n(\vec{x}_{\vec{\kappa}_n})\ \big] \tag{7}$$

2.2 SAMPLING

The training dataset for a Feat2Vec model consists of only semantic observations. In natural language, these would be documents written by humans. Since Structured Deep-In Factorization Machine (Equation 6) requires positive and negative examples, we also need to supply observations that are not semantic. Consider a feature type $\mathcal{F}_i \in \mathbb{R}^{d_i}$, that exists in very high dimensional space (d_i is large). For example, this could happen because we are modeling with one-hot encoding a categorical variable with large number of possible values. In such scenario, it is overwhelmingly costly to feed the model all negative labels, particularly if the model is fairly sparse.

A shortcut around this is a concept known as *implicit sampling*, where instead of using all of the possible negative labels, one simply samples a fixed number (k) from the set of possible negative labels for each positively labelled record. Word2Vec makes use of an algorithm called Negative Sampling (Dyer, 2014). In short, their approach samples a negative observation of a feature w from a noise distribution q:

$$\text{Draw } w \sim p(W|Y=0,X) \tag{8}$$

$$\approx q(\{w_1, w_2, \ldots, w_n\}, \alpha) \tag{9}$$

Here q is the empirical frequency of w in the dataset, exponentiated by α, a flattening hyperparameter:

$$q(W,\alpha) = \frac{c_X(w)^\alpha}{\sum_{w' \in W} c_X(w')^\alpha} \tag{10}$$

Here, $c_X(w)$ is the number of times a feature w appeared in the training dataset X. With $\alpha = 1$, the noise distribution corresponds to the empirical frequency distribution, and with $\alpha = 0$ to the uniform distribution. Intermediate values are intended to be continuous progressions to these two extremes, but in Appendix A.1 we document that this is not necessarily the case.

Our main contribution is introducing a new implicit sampling method that enables learning embedding for structured feature sets. We can learn the correlation of features within a dataset by imputing negative labels, simply by generating "unobserved" records as our negative sample. Unlike Word2Vec, we do not constraint features types to be words. Features types can be individual numeric columns, but they need not to be. By defining feature types as groups of subfeatures (using κ parameter in Equation 6), the model can reason on more complex entities. For example, in our experiments on a movie dataset, we use a "genre" feature type, where we group non-mutually exclusive indicators for comedy, action, and drama films. For more involved feature types (e.g., an image), one would need to define a function ϕ that builds intermediate layers to map the entity to an embedding .

We start with a dataset S^+ of records with n feature types. We then mark all observed records in the training set as positive examples. For each positive record, we generate k negative labels using the following 2-step algorithm:

Algorithm 1 Implicit sampling algorithm for Feat2Vec

function FEAT2VEC_SAMPLE($S^+, k, \alpha_1, \alpha_2$)
  $S^- \leftarrow \emptyset$
  for $s \in S^+$ **do**
    Draw a random feature type $\kappa_i \sim q_1(\{d_i\}_{i=1}^n, \alpha_1)$
    for $j \in \{1, \ldots, k\}$ **do**
      $\vec{e} \leftarrow \vec{x}^{(s)}$ ▷ set initially to be equal to a positive sample
      Draw a random (sub)feature $\vec{r} \sim q_2(\mathrm{X}_{\kappa_i}, \alpha_2)$
      $\vec{e}_{\kappa_i} \leftarrow \vec{r}$ ▷ substitute the i-th feature type with a sampled one
      $S^- \leftarrow S^- + \{\vec{e}\}$
    end for
  end for
  return S^-
end function

Here $q(\mathrm{X}_{\kappa_i}, \alpha_2)$ is the "flattened" empirical multionomial distribution over the frequency of the feature values of feature f_s:

write! (11)

we also need to write the first noise distribution

Often, due to the large # of possible values for a given feature type (i.e. millions of users/movies), 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. Typically one resamples the feature f that define how the negative labels are constructed (i.e. movies a user didn't watch in the Netflix example) from a flattened multionomial distribution :

$$x_j \sim q(\mathbf{X_f}, \alpha), \text{ where } P_q(x_j = x|\mathbf{X_f}, \alpha) \propto \hat{c}_x^{\alpha}, \alpha \in [0, 1]$$

here $\hat{c}_x$ is the empirical frequency of x in the observed dataset $\mathbf{X_f}$, and α is a flattening hyperparameter. This distribution is actually quite general, with $\alpha = 1$ corresponding to the empirical frequency distribution, and $\alpha = 0$ to a uniform distribution across feature values.

Our project extends this application by introducing a new implicit sampling method that allows researchers to learn unsupervised embeddings. We can learn the correlation structure within a dataset without any positive or negative labels, simply by generating "unobserved" records as our negatively labeled sample. We now discuss our method.

We start with a dataset S of records, each with p features. Features need not be individual numeric columns, but instead can be multiple columns that are sensibly grouped together (say, grouping non-mutually exclusive indicators for comedy,action,drame film as a "genre" feature), or even images and text [1]. Often, they are categories, such as user ID, movie ID, book ID, etc. We then mark all observed records as positive, and for each positive record, we generate k negative labels using the following 2-step algorithm:

`for` $s \in S\{$

1. randomly sample one of the features $f_s \sim q_1(\{|\vec{\phi}|_i\}_{i=1}^{p}, \alpha_1)$ to alter for the negative samples, where $q_1(.)$ is a multinomial distribution over features groups in $\vec{\mathcal{F}}$ that selects each feature proportional to its complexity. $|\vec{\phi}|_i$ denotes the number of parameters associated with feature group i, representing the feature's relative complexity. α_1 is a flattening hyperparameter.
2. `for` $(j \in 1, \ldots, k)$
   - Assign the negative label an attribute from noise distribution $x_{s,j} \sim q_2(\mathbf{X_{f_s}}, \alpha_2)$, where $q_2(.)$ is also a "flattened" empirical multionomial distribution over the frequency of the observed feature values of feature f_s, $\mathbf{X_{f_s}}$, and α_2 is once again a flattening hyperparameter.

   }

}

Figure 1: 2-step algorithm for sampling negative label data

Explained in words, our negative sampling method is to randomly select one of the features f from the noise distribution $q_1(.)$, and, conditional on the feature to be resampled, choose another value x_j for feature f from a noise distribution $q_2(.)$. In our application, we use the same class of noise

[1] this requires usage of *deep-in* Factorization Machines, an extension created by Ralph and Jose previously. It would involve building intermediate layers (e.g. convolutional layers) to map a image/text to a vector

distributions (flattened multinomial) for both levels of sampling, but this need not be the case. This method will sometimes by chance generate negatively labeled samples that *do* exist in our sample of observed records. For this reason, among others, we employ a technique known as noise contrastive estimation (Gutmann & Hyvärinen, 2010) which uses basic probability laws to adjust the structural statistical model $p(Y = 1|\vec{\phi}, \vec{x})$ to account for the possibility of random negative labels that appear identical to positively labeled data. An additional burden of this method, however, is we need to learn additional nuisance parameters $Z_{\vec{x}}$ for each unique record type $\vec{x}$ that transform the score function $s(.)$ into a well-behaved probability distribution that integrates to 1. This introduces an astronomical amount of new parameters and greatly increase the complexity of the model. Instead of estimating these, we appeal to the work of (Mnih & Teh, 2012), who show in the context of language models that setting the $Z_{\vec{x}} = 1$ in advance effectively does not change the performance of the model. [2] Written explicitly, the new structural probability model is:

$$\tilde{p}(Y = 1|\vec{\phi}, \vec{x_i}) = \frac{\exp(s(\vec{x}_i, \vec{\phi}))}{\exp(s(\vec{x}_i, \vec{\phi})) + P_q(\vec{x}_i|\alpha_1, \alpha_2)}$$

where $P_q(.)$ denotes the unconditional probability of a a record $\vec{x_i}$ being drawn from our negative sampling algorithm, and $s(x, \vec{\phi})$ is the scoring function for Factorization Machines:

$$s(x, \vec{\phi}) = \sum_{j=1}^{p} \sum_{k=j+1}^{p} \left\langle \phi_j(\vec{x}_j),\ \phi_k(\vec{x}_k) \right\rangle$$

We can show, with relatively little effort,that our 2-step sampler has some interesting theoretical properties. Namely, the embeddings learned under this model will be equivalent to a convex combination of the embeddings learned from p individual Factorization Machines.

2.3 Learning From data

We optimize Noise Contrastive Estimation(Gutmann & Hyvärinen, 2010) loss, which uses basic probability laws to adjust the structural statistical model $p(Y = 1|\vec{\phi}, \vec{x})$ to account for the possibility of random negative labels that appear identical to positively labeled data. An additional burden of this method, however, is we need to learn additional nuisance parameters $Z_{\vec{x}}$ What is this? for each unique record type $\vec{x}$ that transform the score function $s(.)$ what is this? into a well-behaved probability distribution that integrates to 1. This introduces an astronomical amount of new parameters and greatly increase the complexity of the model. Instead of estimating these, we appeal to the work of (Mnih & Teh, 2012), who show in the context of language models that setting the $Z_{\vec{x}} = 1$ in advance effectively does not change the performance of the model.[3] Written explicitly, the new structural probability model is just:

$$p(Y = 1|\vec{\phi}, \vec{x}) = \frac{e^{s(\vec{x}_i, \vec{\phi})})}{e^{s(\vec{x}_i, \vec{\phi})} + q(\vec{x}_i)} \tag{12}$$

We can show, with relatively little effort,that our 2-step sampler has some interesting theoretical properties. Namely, the embeddings learned under this model will be equivalent to a convex combination of the embeddings learned from p individual Factorization Machines.

Theorem 1. *The embeddings learned with* Feat2Vec *are a convex combination of the embeddings learned from* p *individual Factorization Machines.*

[2](Mnih & Teh, 2012) suggests one can set the normalizing constant to 1 because the neural net model has enough free parameters that it will effectively just learn the probabilities itself so that it does not over/under predict the probabilities on average (since that will result in penalties on the loss function). Note that while we follow the same procedure, one could set $Z_{\vec{x}}$ to any positive value in advance and the same logic would follow, if one was worried about astronomically low probabilities.

[3](Mnih & Teh, 2012) suggests one can set the normalizing constant to 1 because the neural net model has enough free parameters that it will effectively just learn the probabilities itself so that it does not over/under predict the probabilities on average (since that will result in penalties on the loss function). Note that while we follow the same procedure, one could set $Z_{\vec{x}}$ to any positive value in advance and the same logic would follow, if one was worried about astronomically low probabilities.

Proof. Let S_f denote the records whose corresponding negative samples resample feature f. We can express the loss function $L(.)$, the binary cross-entropy of the data given the Feat2Vec model, as follows:

Need to use consistent notation in the paper. This notation is really nice:

$$
\begin{aligned}
L(\vec{\mathbf{x}}|\vec{\phi}) =& \frac{1}{|S|}\sum_{i\in S}\Big(\log(\tilde{p}(Y=1|\vec{\phi},\vec{\mathbf{x}}_i)) + \sum_{\vec{\mathbf{x}}_j\sim q(.|\vec{\mathbf{x}}_i)}^{k}\log(\tilde{p}(Y=0|\vec{\phi},\vec{\mathbf{x}}_j))\Big) \\
=& \frac{1}{|S|}\sum_{i\in S}\Big(\log(\tilde{p}(Y=1|\vec{\phi},\vec{\mathbf{x}}_i, i\in S_f)p(i\in S_f)) + \sum_{\vec{\mathbf{x}}_j\sim q(.|\vec{\mathbf{x}}_i)}^{k}\log(\tilde{p}(Y=0|\vec{\phi},\vec{\mathbf{x}}_j, i\in S_f)p(i\in S_f))\Big) \\
=& \frac{1}{|S|}\sum_{f=1}^{p}\sum_{i\in S_f}\Big(\log(\frac{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}p(i\in S_f)}{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}+P_q(\vec{\mathbf{x}}_i|\vec{\mathbf{x}}_i, i\in S_f)}) + \sum_{\vec{\mathbf{x}}_j\sim q(.|\vec{\mathbf{x}}_i, i\in S_f)}^{k}\log(\frac{P_q(\vec{\mathbf{x}}_j|\vec{\mathbf{x}}_i, i\in S_f)p(i\in S_f)}{e^{s(\vec{\mathbf{x}}_j,\vec{\phi})}+P_q(\vec{\mathbf{x}}_j|\vec{\mathbf{x}}_i, i\in S_f)})\Big)
\end{aligned}
$$

Note now that $P_q(\vec{\mathbf{x}}_k|\vec{\mathbf{x}}_i, i\in S_f)$ is simply the probability of the record's feature value $\vec{\mathbf{x}}_{k,f}$ under the second step noise distribution $q_2(\mathbf{X_f}, \alpha_2)$: $P_q(\vec{\mathbf{x}}_k|\vec{\mathbf{x}}_i, i\in S_f) = P_{q_2}(\vec{\mathbf{x}}_{k,f})$, where k refers to record i or a record j negatively sampled from i.

$$
\begin{aligned}
=& \frac{1}{|S|}\sum_{f=1}^{p}\sum_{i\in S_f}\Big(\log(\frac{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}p(i\in S_f)}{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}+P_{q_2}(\vec{\mathbf{x}}_{i,f})}) + \sum_{\vec{\mathbf{x}}_j\sim q(.|\vec{\mathbf{x}}_i, i\in S_f)}^{k}\log(\frac{P_{q_2}(\vec{\mathbf{x}}_{j,f})p(i\in S_f)}{e^{s(\vec{\mathbf{x}}_j,\vec{\phi})}+P_{q_2}(\vec{\mathbf{x}}_{j,f}))}\Big) \\
=& \frac{1}{|S|}\sum_{f=1}^{p}\sum_{i\in S_f}\Big(\log(\frac{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}}{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}+P_{q_2}(\vec{\mathbf{x}}_{i,f})}) + \log(p(i\in S_f)^{k+1}) \\
&+ \sum_{\vec{\mathbf{x}}_j\sim q(.|\vec{\mathbf{x}}_i, i\in S_f)}^{k}\log(\frac{P_{q_2}(\vec{\mathbf{x}}_{j,f})}{e^{s(\vec{\mathbf{x}}_j,\vec{\phi})}+P_{q_2}(\vec{\mathbf{x}}_{j,f})})\Big)
\end{aligned}
$$

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

$$
\begin{aligned}
&\propto \frac{1}{|S|}\sum_{f=1}^{p}\sum_{i\in S_f}\Big(\log(\frac{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}}{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}+P_{q_2}(\vec{\mathbf{x}}_{i,f})}) + \sum_{\vec{\mathbf{x}}_j\sim q(.|\vec{\mathbf{x}}_i, i\in S_f)}^{k}\log(\frac{P_{q_2}(\vec{\mathbf{x}}_{j,f})}{e^{s(\vec{\mathbf{x}}_j,\vec{\phi})}+P_{q_2}(\vec{\mathbf{x}}_{j,f})})\Big) \\
\xrightarrow[|S|\to\infty]{}& \sum_{f=1}^{p}p(i\in S_f)E\Big[\log(\frac{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}}{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}+P_{q_2}(\vec{\mathbf{x}}_{i,f})}) + \sum_{\vec{\mathbf{x}}_j\sim q(.|\vec{\mathbf{x}}_i, i\in S_f)}^{k}\log(\frac{P_{q_2}(\vec{\mathbf{x}}_{j,f})}{e^{s(\vec{\mathbf{x}}_j,\vec{\phi})}+P_{q_2}(\vec{\mathbf{x}}_{j,f})})\Big] \\
&= \sum_{f=1}^{p}p(i\in S_f)E\Big[L(\vec{\mathbf{x}}|\vec{\phi}, \text{target} = f)\Big]
\end{aligned}
$$

Thus, the loss function is just a convex combination of the loss functions of the targeted classifiers for each of the p features, and by extension so is the gradient since:

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

Thus the algorithm will, at each step, learn a convex combination of the gradient for a targeted classifier on feature f, with weights proportional to the feature type sampling probabilities in step 1 of the sampling algorithm.

□

3 LIMITATIONS

Continuous features require a sensible feature function, and future work could focus on them.

-Feature engineering (grouping) is important.

4 EMPIRICAL RESULTS

5 RELATION TO PRIOR WORK

6 CONCLUSION

A APPENDIXES

A.1 PROBLEMS WITH FLATTENING MULTINOMIAL OF Word2Vec

Write

REFERENCES

Anonymized. Structured deep factorization machine: Towards general-purpose architectures. In *6th International Conference on Learning Representations*, Under review.

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

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

Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pp. 2177–2185, 2014.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013b.

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

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

Tobias Schnabel, Igor Labutov, David M Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. In *EMNLP*, pp. 298–307, 2015.