`feat2vec`: Generalized Feature-to-Vector Representations

Luis Armona[*], Jose Gonzalez-Brenes

Stanford University, Chegg

September 6, 2017

Abstract

We implement a novel algorithm using Factorization Machines with implicit sampling to learn general purpose embedding representations of different types of features. Motivated by `word2vec`, this algorithm exploits the structure of a dataframe to more efficiently learn embeddings in a way that is structurally more sensible. This algorithm could be used to create general purpose embeddings for users, products, characteristics, and any other relevant attributes or characteristics that a business or researcher would be interested in representing in a common vector space. Applications include using similarity measures between vectors for reccomender systems, importing embeddings into small-data settings that contain rich external information, and dramatically reducing model complexities by fixing in advance parameters for features we have vector representations of, rather than relearning the parameters for a specific task. Using propietary Chegg Data and a public IMDB dataset, we show that `feat2vec` outperforms `word2vec` both in recommending similar items to a user known from a left-out dataset, and in learning a prediction model for a new feature that we exclude from the training process.

This document serves as an abstract for the research project Jose and Luis embarked on in the summer of 2018. The basic idea is how to create a d-dimensional vector representations ϕ_i of Chegg entities/features x_i that are not targeted towards a specific prediction task. The building block of our model are Factorization Machines [1], which scores the likelihood of an observation being labeled positive / negative $Y \in \{0, 1\}$ as proportional to the sum of all pairwise interactions of p features embeddings $\vec{\phi}$:

$$Pr(Y = 1|\vec{\phi}, \vec{b}, \vec{x}) = \sigma(s(\vec{x}, \vec{b}, \vec{\phi})) = \sigma(b_0 + \sum_{i=1}^{p} b_i x_i + \sum_{i=1}^{p} \sum_{j=i+1}^{p} < \phi_i x_i, \phi_j x_j >)$$

[*]`larmona@stanford.edu`

where $\vec{b}$ is the vector of all additional bias terms for each feature, along with a general intercept term b_o. While bias terms are allowed, we discard them (i.e. set $\vec{b} = \vec{0}$) since we cannot easily transport this inforamtion to other settings. This type of model is typically used to learn latent preferences for products in a market; the archetypal example of this application is learning user preferences of movies on the Netflix Platform[2].

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 samples the feature w 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 $MN(\{w\}, \alpha)$:

$$Pr(W = w|\mathbf{x}, Y = 0) \propto \hat{p}_w^{\alpha}, \alpha \in [0, 1]$$

here $\hat{p}_b$ is the empirical frequency of the w in the dataset, 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 books.

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 negative sample. I now discuss our method.

We start with a dataset S of records 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:

Explained in words, our negative sampling method is to randomly select one of the features, and choose another value for it from a noise distribution $q()$ (here, the empirical flattened multinomial distribution is our noise distribution). Of course, this method will sometimes by chance generate observations with negative labels that *do* in fact exist in our sample of observed records. For this reason, among others, we employ a technique known as noise contrastive estimation[5] which uses basic probability laws to adjust the structural statistical model $Pr(Y = 1|\vec{\phi}, \vec{x})$ to account for the possibility of random negative labels that appear identical to positively labeled data. An additional burden

[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

```
for $s \in S${
    1. randomly choose one of the features to alter for the negative samples from a multinomial over the feature types $f_s \sim \mathrm{MN}(\{|\vec{\phi}|_i\}_{i=1}^{p}, \alpha_1)$, where $|\vec{\phi}|_i$ denotes the number of parameters associated with feature $i$, representing the feature's relative complexity.
    2. for $(j \in 1, \ldots, k)$
        • Assign the negative label an attribute from noise distribution $x_{s,j} \sim \mathrm{MN}(\mathrm{X}_{f_s}, \alpha)$, where $\mathrm{MN}(\mathrm{X}_{f_s}, \alpha_2)$ is the "flattened" empirical multionomial distribution over the frequency of the feature values of feature $f_s$.
    }
}
```

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

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 [4], 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 just:

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

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. We show this in a short proof below. 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:

[2][4] 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.

$$
\begin{aligned}
L(\vec{\mathbf{x}}|\vec{\phi}) &= \frac{1}{|S|}\sum_{i\in S}\Big(\log(\tilde{p}(Y=1|\vec{\phi},\vec{\mathbf{x_i}})) + \sum_{\vec{\mathbf{x}}_j\sim q(\vec{\mathbf{x_i}})}^{k}\log(\tilde{p}(Y=0|\vec{\phi},\vec{\mathbf{x_j}}))\Big)\\
&= \frac{1}{|S|}\sum_{i\in S}\Big(\log(\tilde{p}(Y=1|\vec{\phi},\vec{\mathbf{x_i}}, i\in S_f)p(i\in S_f)) + \sum_{\vec{\mathbf{x}}_j\sim q(\vec{\mathbf{x_i}})}^{k}\log(\tilde{p}(Y=0|\vec{\phi},\vec{\mathbf{x_j}}, i\in S_f)p(i\in S_f))\Big)\\
&= \frac{1}{|S|}\sum_{f=1}^{p}\sum_{i\in S_f}\Big(\log(\frac{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}p(i\in S_f)}{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}+q(\vec{\mathbf{x}}_i|i\in S_f)}) + \sum_{\vec{\mathbf{x}}_j\sim q(\vec{\mathbf{x_i}}|i\in S_f)}^{k}\log(\frac{q(\vec{\mathbf{x}}_j|i\in S_f)p(i\in S_f)}{e^{s(\vec{\mathbf{x}}_j,\vec{\phi})}+q(\vec{\mathbf{x}}_j|i\in S_f)})\Big)\\
&= \frac{1}{|S|}\sum_{f=1}^{p}\sum_{i\in S_f}\Big(\log(\frac{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}}{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}+q(\vec{\mathbf{x}}_i|i\in S_f)}) + \log(p(i\in S_f)^{k+1})\\
&+ \sum_{\vec{\mathbf{x}}_j\sim q(\vec{\mathbf{x_i}}|i\in S_f)}^{k}\log(\frac{q(\vec{\mathbf{x}}_j|i\in S_f)}{e^{s(\vec{\mathbf{x}}_j,\vec{\phi})}+q(\vec{\mathbf{x}}_j|i\in S_f)})\Big)\\
&\propto \frac{1}{|S|}\sum_{f=1}^{p}\sum_{i\in S_f}\Big(\log(\frac{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}}{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}+q(x_{i,f})}) + \sum_{\vec{\mathbf{x}}_j\sim q(\vec{\mathbf{x_i}}|i\in S_f)}^{k}\log(\frac{q(x_{j,f})}{e^{s(\vec{\mathbf{x}}_j,\vec{\phi})}+q(x_{j,f})})\Big)\\
&\xrightarrow[|S|\to\infty]{} \sum_{f=1}^{p}p(i\in S_f)E\Big[\log(\frac{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}}{e^{s(\vec{\mathbf{x}}_i,\vec{\phi})}+q(x_{i,f})}) + \sum_{\vec{\mathbf{x}}_j\sim q(\vec{\mathbf{x_i}}|i\in S_f)}^{k}\log(\frac{q(x_{j,f})}{e^{s(\vec{\mathbf{x}}_j,\vec{\phi})}+q(x_{j,f})})\Big]\\
&= \sum_{f=1}^{p}p(i\in S_f)E\Big[L(\vec{\mathbf{x}}|\vec{\phi},\text{target}=f)\Big]
\end{aligned}
$$

where the third to last line drops the probability of assignment to a feature type term since it is outside of the model parameters $\vec{\phi}$ and fixed in advance. 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. Thus it will learn a convex combination of the embeddings for each classifier, with weights proportional to the feature type sampling probabilities in step 1. of the sampling algorithm.

References

[1] Rendle, Steffen. "Factorization machines." Data Mining (ICDM), 2010 IEEE 10th International Conference on. IEEE, 2010.

[2] Koren, Yehuda, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." Computer 42.8 (2009).

[3] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient Estimation of Word Representations in Vector Space." In Proceedings of Workshop at ICLR, 2013.

[4] Andriy Mnih and Yee Whye Teh. "A fast and simple algorithm for training neural probabilistic language models". In Proceedings of the 29th International Conference on Machine Learning, 2012.

[5] Gutmann, Michael, and Aapo Hyvrinen. "Noise-contrastive estimation: A new estimation principle for unnormalized statistical models." Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. 2010.