Categorical Embeddings for Tabular Data using PyTorch

Sanskruti Khedkar[1*], *Shilpa Lambor*[2], *Yogita Narule*[3], *Prathamesh Berad*[4]

[1]Multidisciplinary Engineering Department, Vishwakarma Institute of Technology, Pune, 411037, Maharashtra, India
[2]Multidisciplinary Engineering Department, Vishwakarma Institute of Technology, Pune, 411037, Maharashtra, India
[3]Multidisciplinary Engineering Department, Vishwakarma Institute of Technology, Pune, 411037, Maharashtra, India
[4]Multidisciplinary Engineering Department, Vishwakarma Institute of Technology, Pune, 411037, Maharashtra, India

Abstract. Deep learning has received much attention for computer vision and natural language processing, but less for tabular data, which is the most prevalent type of data used in industry. Embeddings offer a solution by representing categorical variables as continuous vectors in lowdimensional space. PyTorch provides excellent support for GPU acceleration and pre-built functions and modules, making it easier to work with embeddings and categorical variables. In this research paper, we apply a feedforward neural network model in PyTorch to a multiclass classification problem using the Shelter Animal Outcome dataset. We calculate the probability of an animal's outcome belonging to each of the 5 categories. Additionally, we explore feature importance using two common techniques: MDI and permutation. Understanding feature importance is crucial for building better models, improving performance, and interpreting and communicating results. Our findings demonstrate the usefulness of embeddings and PyTorch for deep learning with tabular data and highlight the importance of feature selection for building effective machine learning models.

1 Introduction

Use shelters play an important role in ensuring the welfare of animals and providing them with temporary care until they are adopted. However, managing an animal shelter involves numerous challenges, including limited resources, overcrowding, and a high volume of incoming animals. As such, animal shelters are increasingly turning to machine learning as a tool for predicting animal shelter outcomes, which can help improve the allocation of resources, reduce overcrowding, and ultimately improve animal welfare.

* Corresponding author: sanskruti.khedkar21@vit.edu

It is important to use encoding techniques to convert the category variables into numerical values. In this paper [1] author M. K. Dahouda and I. Joe proposed, a method for encoding categorical features on categorical datasets using deep learning. This involves creating a distributed representation for each category by training a neural network to discover the characteristics of each vector. To do this, a data vocabulary of categorical data is established and each category is turned into a single word using word tokenization. The category information is then mapped to word vectors using feature learning. Results show that this deep-learned embedding method outperforms traditional one-hot encoding with an F1 score of 89% compared to 71%. Additionally, it requires less memory and produces fewer features. On numerous well-known classification benchmark data sets, pretrained categorical embeddings outperform one-hot encoding. The GloVe embedding technique provided the most reliable improvements. Every time, adjusted unsupervised embeddings performed better than other embedding techniques.[2]

To get the best categorization outcomes possible, numerous machine learning algorithms were trained. When comparing the results of different algorithms in the research, it was found that the K-Nearest Neighbors and C4.5 algorithms performed the best in terms of classification accuracy. Because the dataset in this study [3] by K. Mitrović, D. Milošević and M. Greconici, has a large number of Categorical Embeddings for Tabular Data using PyTorch Yogita Narule, Shilpa Lambor, Sanskruti Khedkar, Prathamesh Berad Department of Multidisciplinary Engineering, Vishwakarma Institute of Technology, Pune, 411037, Maharashtra, India A instances while having a low dimensionality, the straightforward methods produced good results. The SMOTE technique for handling imbalanced data improves all metrics as well as accuracy. SMOTE and KNN algorithms produced the greatest results (80.7139%).

A new deep learning library called PyTorch Tabular makes it simple and quick to work with tabular data and deep learning. This library works directly with pandas dataframes and is developed on top of PyTorch and PyTorch Lightning. In order to free you up to concentrate on the model, PyTorch Tabular makes an effort to make the "software engineering" aspect of working with neural networks as simple and painless as possible. It also aims to combine the various Tabular innovations into a single framework with an uniform API that will function with various cutting-edge models.[4] The use of latent component models allows for the representation of categorical features through a low-dimensional approach, providing an advantageous perspective. To combine the benefits of latent factor embedding and tree components while avoiding their respective drawbacks, a novel model named GB-CENT has been introduced. In many practical scenarios, datasets often contain a combination of numerous numerical features as well as categorical features with a high cardinality, such as geolocations, IDs, and tags. As proposed by Qian Zhao, Yue Shi, and Liangjie Hong in the paper [5], that when tested on two real-world datasets, GB-CENT outperformed contemporary matrix factorization models and decision tree-based models, as well as their ensembles, in terms of accuracy. GB-CENT achieved this superior accuracy in an efficient manner, with both speed and precision being notable advantages of the model. The GB-CENT model outperforms the GBDT and state-of-the-art matrix factorization models by a wide margin. Moreover, it performs better than the feature level ensemble of the two models. The paper [6] suggests a brand-new embedding system called Deep Hash Embedding (DHE), which first uses several hashing methods and transformations to encode the feature value into a distinct identifier vector. The identifier vector is transformed into an embedding by a deep neural network (DNN) following the encoding module. Since the encoding module is deterministic and cannot be learned, there is no requirement for parameter storage. The computing of the embedding network is the DHE's bottleneck, albeit it might be sped up with more potent hardware and NN acceleration

techniques like pruning. The implicit and potentially changeable embedding capacity for each feature value in DHE.

Entity embedding has advantages over one-hot encoding such as reduced memory usage and faster neural network performance. More importantly, entity embedding reveals the inherent characteristics of categorical variables by grouping similar values together in the embedding space through clustering. Entity embedding can be used to visualise categorical data and for data clustering because it defines a distance metric for categorical variables. When the input is sparse and the statistics are uncertain, entity embedding aids the neural network in generalizing more effectively. Because other approaches have a tendency to overfit, it is particularly helpful for datasets with a lot of high cardinality characteristics. [7]. In this study [8] authors propose PyTorch, a Python library. PyTorch performs dynamic tensor calculations instantly, with automatic differentiation and GPU acceleration, and does so with performance similar to the fastest current methods. deep learning reference materials. In the scientific community, this pairing has shown to be quite well-liked. S. Kim, H. Wimmer and J. Kim [9] highlights the gap in performance between convolutional neural networks (CNNs) and other deep learning models while using various artificial neural library implementations. For each of the three CNN models, libraries like Keras, Pytorch, and MXnet were used. Binary image classification was then carried out using the Dogs vs. Cats dataset from Kaggle. Each CNN model provided a distinct F1 score value and accuracy because 25% of the dataset was used as a testing set, while the remaining 75% was used as a training set. The Austin Animal Center's intake and outcome data from 2013 to July 2022 are used in this study[10]. First, a single table with the intake and outcome data is created. The adoption times are then split into two main categories: less than or greater than 15 days to adopt. A tree-based boosting method is used to quantify the influence of various variables. Using the necessary splits of the dataset, this approach is trained, validated, and tested. The writers present a thorough examination of the factors influencing various stay periods at the conclusion.

In this research paper, we explore the use of categorical embeddings, random forests, and permutation methods in predicting animal shelter outcomes using the Shelter Animal Outcome dataset available on Kaggle. The main goal is to create a reliable predictive model that can precisely estimate the likelihood of an animal's outcome falling into any of the five categories present in the dataset: adoption, transfer, return to owner, euthanasia, or death.

2 Methodology

First the dataset exploration, data pre-processing is done followed by label encoding and most important step categorical embedding. Then feed forward neural network model is trained and the test output show the probability of particular animal falling in each of the five outcomes. Further the feature importance is implemented using two methods, one the random forest feature importance and the other one permutation feature importance.

2.1 Dataset and Data Pre-processing

The Animal Shelter Outcomes dataset available on Kaggle is used for developing a predictive model that can accurately predict the probability of an animal's outcome belonging to each of the five categories in the dataset: adoption, transfer, return to owner, euthanasia, and death. The dataset contains information about the outcomes of animals in the care of the Austin Animal Center in Austin, Texas, USA. The dataset includes information about the breed, color, sex, age, and size of the animals, as well as the date and time they were admitted to the

shelter and their outcome. The first step in data pre-processing is to remove NA values. Next, the AnimalID column is eliminated because it is distinct and useless for training. The Outcome Subtype column is eliminated because, although it is a component of the objective, we are not required to predict it. The Date-Time column is eliminated because it didn't appear vital to have the precise timestamp of when the entry was entered. Due to the excess of NA values, the Name column was removed (more than 10k missing). It also didn't seem to be a crucial element in deciding an animal's destiny.

2.2 Label Encoding and Categorical Embeddings

Since the model will take only numeric input, it is necessary to convert all categorical elements to numbers. Label Encoder class from the scikit-learn library is used to encode the categorical columns.

	AnimalType	SexuponOutcome	AgeuponOutcome	Breed	Color
0	1	3	5	1482	146
1	0	4	5	775	184
2	1	3	21	1293	97
3	0	1	26	775	47
4	1	3	21	1101	311

Fig. 1

Figure 1 shows the dataset after performing data pre-processing and Label Encoding. Label Encoding has converted the labels into integer for example in 'AnimalType' column the 'Dog' is labelled as '1' and 'Cat' is labelled as '0'.

2.3 Model and Training

The Pytorch Dataset class simplifies dataset access during training and facilitates efficient batch management using the Data Loader module. To embed only categorical columns, we divide the input into two sections: numerical and categorical. We select a batch size and pass it, along with the dataset, to the Data Loader. Deep learning typically operates on batches, and the Data Loader streamlines batch management by shuffling data before training.

```
class ShelterOutcomeDataset(Dataset):
    def __init__(self, X, Y, embedded_col_names):
        X = X.copy()
        self.X1 = X.loc[:,embedded_col_names].copy().values.astype(np.int64) #categorical columns
        self.X2 = X.drop(columns=embedded_col_names).copy().values.astype(np.float32) #numerical columns
        self.y = Y

    def __len__(self):
        return len(self.y)

    def __getitem__(self, idx):
        return self.X1[idx], self.X2[idx], self.y[idx]
```

Fig. 2

In figure 2 ShelterOutcomeDataset class is shown where two main methods ‘len’ and ‘getitem’ are overwritten.

The data is segregated into two parts: continuous and categorical. The categorical part is transformed into embedding vectors according to predetermined sizes and concatenated with the continuous part. This concatenated data is then fed into the rest of the network.

```
class ShelterOutcomeModel(nn.Module):
    def __init__(self, embedding_sizes, n_cont):
        super().__init__()
        self.embeddings = nn.ModuleList([nn.Embedding(categories, size) for categories,size in embedding_sizes])
        n_emb = sum(e.embedding_dim for e in self.embeddings)
        self.n_emb, self.n_cont = n_emb, n_cont
        self.lin1 = nn.Linear(self.n_emb + self.n_cont, 200)
        self.lin2 = nn.Linear(200, 70)
        self.lin3 = nn.Linear(70, 5)
        self.bn1 = nn.BatchNorm1d(self.n_cont)
        self.bn2 = nn.BatchNorm1d(200)
        self.bn3 = nn.BatchNorm1d(70)
        self.emb_drop = nn.Dropout(0.6)
        self.drops = nn.Dropout(0.3)


    def forward(self, x_cat, x_cont):
        x = [e(x_cat[:,i]) for i,e in enumerate(self.embeddings)]
        x = torch.cat(x, 1)
        x = self.emb_drop(x)
        x2 = self.bn1(x_cont)
        x = torch.cat([x, x2], 1)
        x = F.relu(self.lin1(x))
        x = self.drops(x)
        x = self.bn2(x)
        x = F.relu(self.lin2(x))
        x = self.drops(x)
        x = self.bn3(x)
        x = self.lin3(x)
        return x
```

Fig 3

In figure 3, embeddings using nn.Embedding from PyTorch is used. Also the main model i.e feed forward neural network is shown which takes categorical and continuous train data as input.

The Adam optimizer is used to optimize the cross-entropy loss. The training is pretty straightforward: iterate through each batch, do a forward pass is executed, gradients are calculated, and a gradient descent is performed. This sequence is repeated for as many epochs as required.

2.4 Test Output

To generate class probabilities for test inputs, we applied the Softmax function to our model's output.

```
training loss:  1.198108796507249
valid loss 0.960 and accuracy 0.599
training loss:  1.0223988591930895
valid loss 0.922 and accuracy 0.606
training loss:  0.988031951363078
valid loss 0.910 and accuracy 0.617
training loss:  0.9689590934110235
valid loss 0.890 and accuracy 0.639
training loss:  0.9579916649931961
valid loss 0.876 and accuracy 0.635
training loss:  0.9535581100847303
valid loss 0.882 and accuracy 0.631
training loss:  0.9516557797586399
valid loss 0.866 and accuracy 0.627
training loss:  0.9468583127252675
valid loss 0.868 and accuracy 0.642
```

Fig. 4

In figure 4, training loop of 8 epochs gives the training loss, valid loss and accuracy for the model. As we can see in the figure the value of training loss helps us to know how well the data was fitted on the dataset. Valid loss data is helping us predict how well the model has generalized new data. Accuracy helps in analysing how well the model predicts the correct output for a given input.

2.5 Feature Importance

To determine the feature importance, we first mapped the Outcome Type categories to numeric labels for each target variable. We then created target variables for each outcome type and split the data into training and test sets for each target variable.

Random Forest classifier is trained for each target variable and feature importance are computed for each target variable. Finally, feature importance for each target variable are visualized.

```
rf_1 = RandomForestClassifier(n_estimators=100, random_state=42)
rf_1.fit(X_train1, y_train1)
rf_2 = RandomForestClassifier(n_estimators=100, random_state=42)
rf_2.fit(X_train2, y_train2)
rf_3 = RandomForestClassifier(n_estimators=100, random_state=42)
rf_3.fit(X_train3, y_train3)
rf_4 = RandomForestClassifier(n_estimators=100, random_state=42)
rf_4.fit(X_train4, y_train4)
rf_5 = RandomForestClassifier(n_estimators=100, random_state=42)
rf_5.fit(X_train5, y_train5)
```

Fig. 5

In figure 5, five separate random forest classifiers are being created and trained using different subsets of data (X_train1 to X_train5 and y_train1 to y_train5). Each classifier has

100 decision trees (n_estimators=100) and is initialized with the same random seed (random_state=42) to ensure reproducibility.

The process involved randomly shuffling each feature column and measuring how much the shuffling reduced the model's accuracy. The greater the reduction in accuracy, the more important the feature.

```
result1 = permutation_importance(rf_1, X_test1, y_test1, n_repeats=10, random_state=42)
result2 = permutation_importance(rf_2, X_test2, y_test2, n_repeats=10, random_state=42)
result3 = permutation_importance(rf_3, X_test3, y_test3, n_repeats=10, random_state=42)
result4 = permutation_importance(rf_4, X_test4, y_test4, n_repeats=10, random_state=42)
result5 = permutation_importance(rf_5, X_test5, y_test5, n_repeats=10, random_state=42)
```

Fig. 6

In figure 6, permutation importance is being calculated for each of the five random forest classifiers. It involves randomly permuting the values of a single feature in the test data and measuring the impact on the model's accuracy. This process is repeated multiple times (n_repeats=10) and the average impact on accuracy is calculated.

3 Results and discussions

Results show the percentage of a particular animal ending up in the five target variables and importance of given input features for five target variables. Animal's outcome as 'Died' has less probability as compared to other outcomes. After this, outcomes as 'Euthanasia' has very less probability. Animal either getting adopted or transferred has equal probability in general. In feature importance 'AnimalType' feature has very less importance and 'SexuponOutcome' feature has very high importance in general.

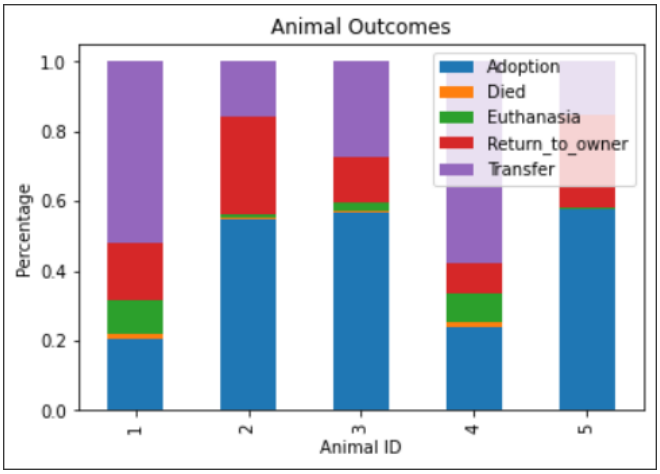

Fig. 7

In figure 7, percentage of particular animal falling in each of the five category is given. For example, for 'AnimalID-1' percentage of animal getting 'Adoption' is shown in blue colour with '20.2%',
'Died' in orange colour with '1.5%', 'Euthanasia' in green with '9.6%', 'Return to owner' in red with '16.6%' and 'Transfer' in purple with '51.8%'.

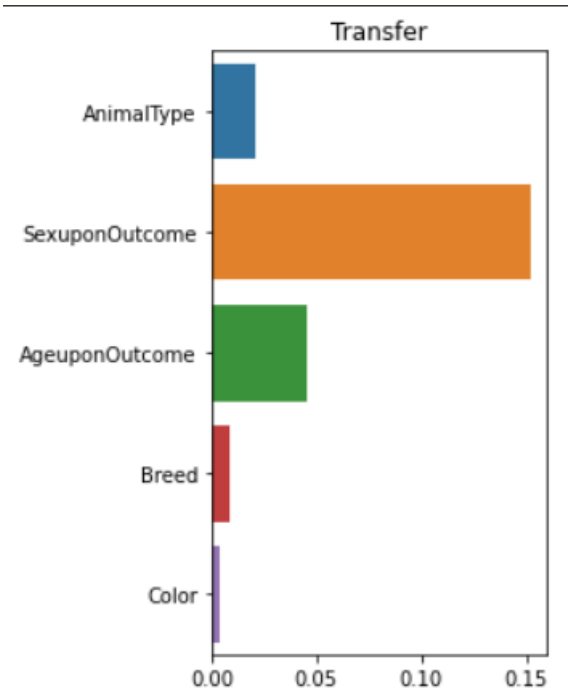

Fig. 8

In figure 8, percentage of importance of input features using the permutation importance. For example, for the 'Transfer' outcome, importance of 'AnimalType' feature is approximately '2.5%', 'SexuponOutcome' feature is approximately '15%', 'AgeuponOutcome' feature is approximately '4.85%', 'Breed' is approximately '1.25%' and 'Color' is approximately '0.98'.

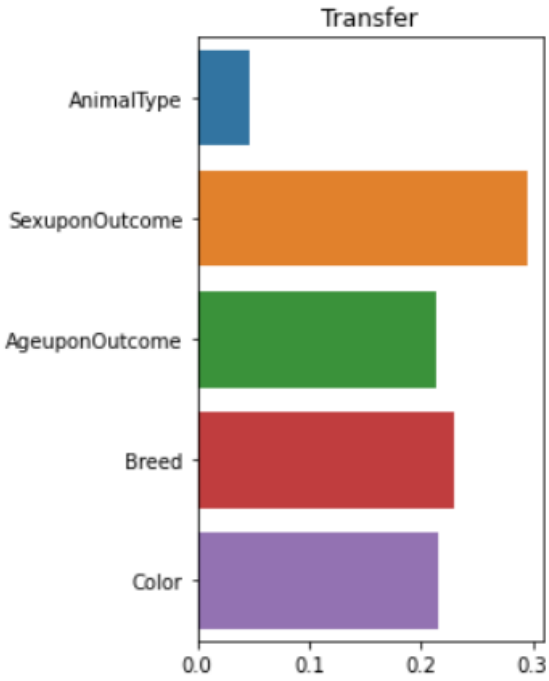

Fig. 9

In figure 9, percentage of importance of input features using the random forest feature importance. For example, for the 'Transfer' outcome, importance of 'AnimalType' feature is approximately '5.28%', 'SexuponOutcome' feature is approximately '28.7%', 'AgeuponOutcome' feature is approximately '21.1%', 'Breed' is approximately '24.3%' and 'Color' is approximately '22%'.

4 Conclusion and Future Scope

The use of deep learning for tabular data, especially in the context of categorical variables, has gained significant popularity in both research and industry communities. To optimize the performance of deep learning models on such datasets, it is crucial to employ efficient ways to use embeddings for categorical variables. Embeddings have proven to be highly beneficial as they save memory and reveal intrinsic properties of categorical variables. Additionally, when it comes to feature importance techniques, permutation importance has shown to provide more accurate estimates of feature importance in datasets with highly correlated or redundant features. Random forest feature importance, on the other hand, tends to overestimate the importance of highly correlated features. Therefore, using permutation importance can provide more reliable insights into feature importance and help to avoid potential biases that may arise from highly correlated features. Overall, incorporating embeddings and using permutation importance can help to optimize the performance and accuracy of deep learning models for tabular data with categorical variables.

While our study demonstrates the usefulness of embeddings and PyTorch for deep learning with tabular data, there is scope for further research in this area. One avenue for

future research is to explore more advanced deep learning architectures beyond the basic feedforward neural network used in our study. Further research in this area could explore the potential of PyTorch for optimizing and accelerating the training process of deep learning models with tabular data, and for developing more advanced deep learning architectures that can better capture the complex relationships between input features and output labels.

References

1. M. K. Dahouda and I. Joe, "A Deep-Learned Embedding Technique for Categorical Features Encoding," in IEEE Access, vol. 9, pp. 114381- 114391, 2021, doi: 10.1109/ACCESS.2021.3104357
2. H. De Meulemeester and B. De Moor, "Unsupervised Embeddings for Categorical Variables," 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, UK, 2020, pp. 1-8, doi: 10.1109/IJCNN48605.2020.9207703.
3. K. Mitrović, D. Milošević and M. Greconici, "Comparison of Machine Learning Algorithms for Shelter Animal Classification," 2019 IEEE 13th International Symposium on Applied Computational Intelligence and Informatics (SACI), Timisoara, Romania, 2019, pp. 211-216, doi: 10.1109/SACI46893.2019.9111575.
4. Joseph, Manu. "Pytorch tabular: A framework for deep learning with tabular data." arXiv preprint arXiv:2104.13638 (2021).
5. Qian Zhao, Yue Shi, and Liangjie Hong. 2017. GBCENT: Gradient Boosted Categorical Embedding and Numerical Trees. In Proceedings of the 26th International Conference on World Wide Web (WWW '17). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 1311–1319.
6. Kang, Wang-Cheng, Derek Zhiyuan Cheng, Tiansheng Yao, Xinyang Yi, Ting Chen, Lichan Hong, and Ed H. Chi. "Learning to embed categorical features without embedding tables for recommendation." arXiv preprint arXiv:2010.10784 (2020).
7. Guo, Cheng, and Felix Berkhahn. "Entity embeddings of categorical variables." arXiv preprint arXiv:1604.06737 (2016).
8. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), Advances in Neural Information Processing Systems 32 (NeurIPS 2019)
9. S. Kim, H. Wimmer and J. Kim, "Analysis of Deep Learning Libraries: Keras, PyTorch, and MXnet," 2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA), Las Vegas, NV, USA, 2022, pp. 54-62, doi: 10.1109/SERA54885.2022.9806734.
10. C. Sazara and X. Gao, "Predicting Animal Shelter Pet Adoption Times and Feature Importance Analysis using CatBoost," 2022 IEEE 11th International Conference on Intelligent Systems (IS), Warsaw, Poland, 2022, pp. 1-4, doi: 10.1109/IS57118.2022.10019608