Experiment Brief: Static Embeddings fine grained

Short Description/TLDR

Top-k class Embedding and Feature Maps are the inputs to produce the joint embedding and the attention map in the fine-grained model. This experiment focuses on finding a method to transfer our taxonomy knowledge to Top-K class Embedding.

Table of Contents *

** When finished, click on the table of contents and hit the refresh button at the top left. This will auto-update the page numbers and any new headings*

Introduction

In our previous approach, the Top-K embedding layer was generated randomly, and it would change over time as it was treated as a learnable value. We hypothesized that by the static embedding approach to encoding knowledge about the relationships between species, we could improve the overall performance of the second prediction task. Specifically, this could reduce the misclassification of species that are similar to each other. This experiment is relevant because it aligns with our group's aim to enhance model accuracy by incorporating domain-specific knowledge into embeddings.
In this experiment, I focused on one type of embedding suggested and explored the results.

Methodology

Datasets

The dataset used for this experiment is called filtered_and_renumbered.csv. This data is ORSAC cleaned but not all images are Entomologist cleaned. This data set has 32 classes and y is renumbered to match the index of mosquito_static_classes.pkl file. In mosquito_static_classes.pkl for species that don't have subgenus, we use species name+sg instead of pad for better results. For instance, Culicini Culex coronatorsg coronator since it doesn't have a subgenus we use species name+sg as the subgenus name.

Algorithms, Models, and Training Methodology

Step 1: Generating Static Embedding

Collab: Static_Embeddings.ipynb

.py: python file can be found in core repository staticembeeding.py

The inputs to generate static embeddings are the taxonomy level, Species Name corresponding to the taxonomy level, and the dimension of each word in the species name. For instance, for the species name "Aedini aedes Stegomyia aegypti", the level of taxonomy is ["Tribe", "Genus", "Subgenus", "Species"] and each word in the species name may be set to 50dim meaning the species name embedding will have 200 dim(50x4 as this is a 4 word in a sentence structure). Given the input, the `static_embedding` method assigns an integer to each word. Then it is reshaped to the desired dimension using Gaussian Random Projection. Then *Sentences_to_indices* concatenates each word to a species/class embedding. Static embedding can generate a word embedding or class embedding. The type of embedding used depends on the implementation.

Step 2: Integrating Static Embedding

There were two methods tested when implementing static embedding. The first method is giving the static embedding as an initial weight so it can learn through the training and the second method is keeping the static embedding constant through training.

For the first method, LSTM was used to process the embedding and the input is word embedding, not class embeddings. Three saved files were needed to provide the static embedding as initial weight: `mosquito_static_300d.npy`, `mosquito_static_classes.pkl`, `mosquito_static_dictionary.pkl`.

a. `mosquito_static_300d.npy`: contains embedding of each word (300 is dim of each words em)
b. `mosquito_static_classes.pkl`: is a list of the species names
c. `mosquito_static_dictionary.pkl`: is a dictionary of word-to-index mapping that corresponds to the index of the embedding and list of each word.

These files are saved in the data directory in the fine-grained repo. To change files for different experiments you need to change emb_dim and emb_mode in TrainConfig, train.py (line: 257), train_utils.py (95,96).

```python
    #train.py
    #mosquito_glove_50d.npy is array of static embedding
     _, _, word2vec = read_glove_vecs('data/mosquito_glove_50d.npy',
'data/mosquito_dictionary.pkl')
  nets = train_config.build_model(config, device, word2vec,
num_classes=num_classes)

#tain_utils.py
#number of classes is 32
self.classes = pickle.load(open(f'data/mosquito_classes.pkl', 'rb'))
self.word_to_index, self.index_to_word, _
=read_glove_vecs(f'data/mosquito_glove_50d.npy',
f'data/mosquito_dictionary.pkl')
indices = sentences_to_indices(self.classes, self.word_to_index, self.max_len)

#TrainConfig
"emb_model": "lstm",
"emb_dim": 300,
```

In the second method, we developed a new class similar to SimpleEmbedding and removed the embedding layer that introduces learning parameters. The goal is for the embeddings to not update during training and the output of the forward method matches the embeddings loaded from the .npy file exactly. The class uses self.register_buffer('embed', word2vec) to load the embedding. The files we provide are `mosquito_static_classes.pkl and mosquito_static_class_768d.npy.`

a. `Mosquito_static_class_768d.npy:` contains the class embedding of each species name.

The embedding is loaded in the class it is possible to change the embedding file in embedding.py class `SimpleEmbedding.` It is also necessary to change the emb_mode in TrainConfig to “static”.

Analysis method

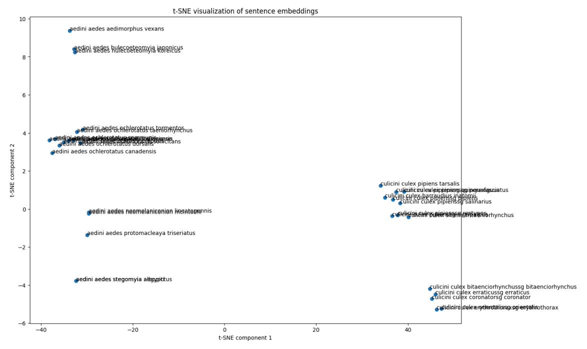
In step 1, the analysis method was a visualization of the species names using TSNE and UMAP. In addition, Silhouette Score and SubGenus Error were used to measure the similarities and differences between species given the embedding. For step 2, the focus was analyzing overall model performance by changing the embedding used. I compared the value of accuaacy_image_last and confusion matrix between the Simple/random, Class/Lstm, and Static embedding. The goal is to achieve the highest accuracy.

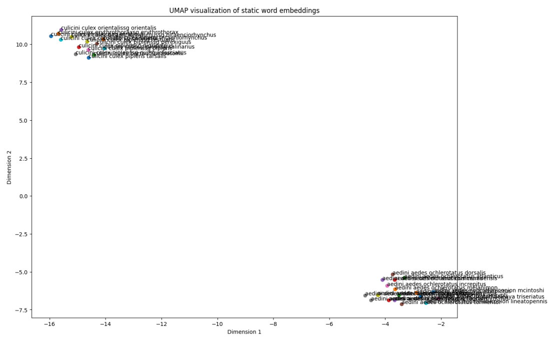
Computing details

I used vectech1, each model was trained on 40 epochs with a batch size of 64. On average it took 3 hours to train the model.

Results and Discussion

The first method was to asses the embedding was visualizing it. In the visualization, we expect that different genera or Tribe are far from each other and similar species are close to each other. Below is a TSNE and UMAP visualization of static embedding(word dim:192 class dim: 768) The decision to use both visualization methods is because TSNE is sometimes incaccurate but umap has a stable result.
You can find visualization of different dim here: Visualization





Silhouette score measures how similar a sample is to its cluster compared to others, providing a measure of clustering quality. Our Silhouette Score started at 0.52 but with the improvement of static embedding algorithm and reshaping process, it has increased to 0.97. This means when the static embedding is clustered using kmeans, each sample is highly similar to its own cluster in the genus level. Then to assess the performance at the subgenus level, I used SubGenus Error. It performs K-Means clustering on sentence embeddings to classify them into 13 clusters, corresponding to the number of subgenus in the 32-class vocabulary. The classification accuracy is evaluated based on how well the clusters align with the true subgenus labels. The subgenus is classified accurately 96% of the time. The results show that the static embedding represents the similarities and differences between species.

Experiments	Test_last	Val_last	Train_last
SimpleEmbedding	94.5221	94.7483	96.774
ClassEmbedding (LSTM)	81.6669	~89	90.94
Static 1	93.8785	94.2767	95.506
Static 2	95.3539	95.6801	97.5923

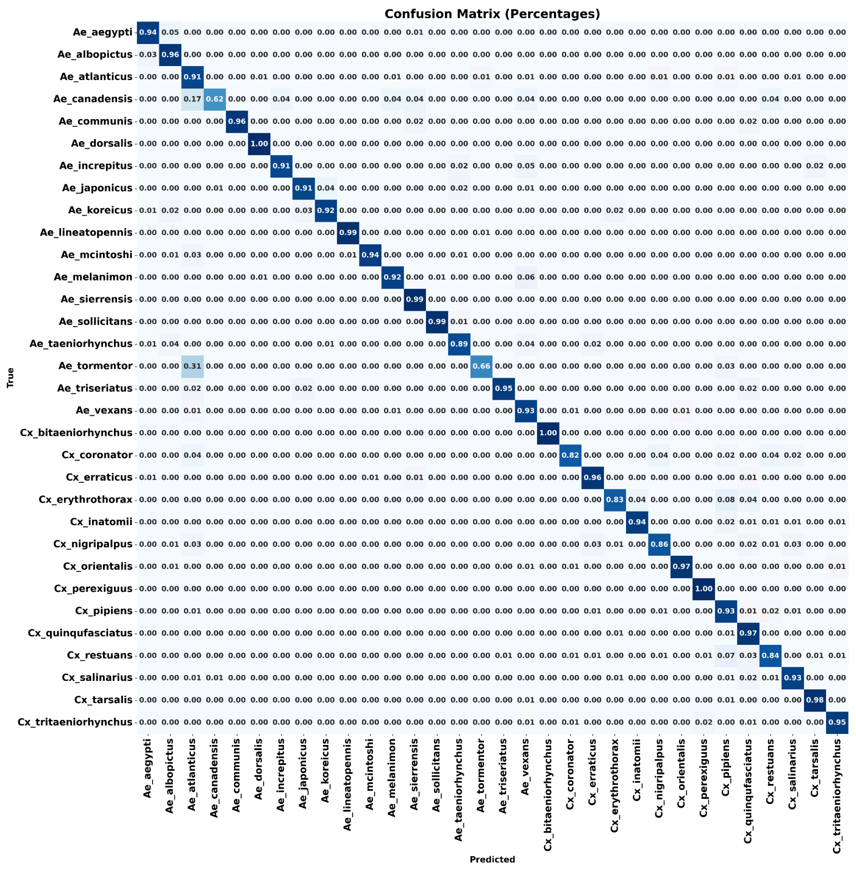
Base Model			

Table1: Accuracy image last of test, validation, and train

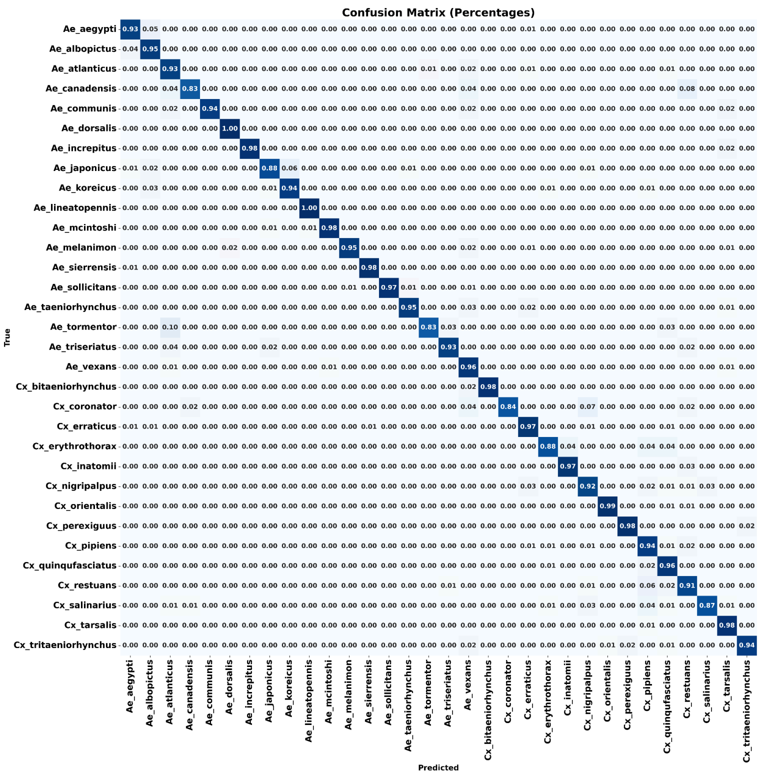
After integrating different types of embeddings, the accuracy results are as follows. In embedding.py, three classes—SimpleEmbedding, StaticEmbedding, and ClassEmbedding (LSTM)—were evaluated. The SimpleEmbedding class serves as the baseline method, with embeddings initialized with random values and updated during training. Among the methods, the LSTM embedding (ClassEmbedding) showed the lowest accuracy. Static 1 refers to the results from the first static embedding class I developed using the nn.Embedding class, while Static 2 refers to the second static embedding class that doesn't utilize nn.Embedding. Notably, the second static embedding class achieved approximately 1% better accuracy compared to the baseline SimpleEmbedding. This improvement indicates that incorporating domain-specific knowledge into the embeddings enhances accuracy. The confusion matrix also highlights this improvement, particularly in the classification of mosquitoes like Ae_canadensis, Ae_tormentor, and Cx_erythrothorax. For instance, the accuracy for Ae_canadensis increased from 62% with the simple embedding to 83%, and for Ae_tormentor, accuracy improved from 66% to 83%.

Confusion matrix-

Simple Embedding



Static embedding



Conclusions

We learned that using static embeddings, or embeddings that incorporate knowledge of species relationships, significantly improves the classification accuracy of mosquitoes. This finding suggests that our repository should adopt static embeddings as the default method for this task. Moving forward, we should experiment with different embedding dimensions, as visualizations show that varying dimensions can lead to slightly different results. Additionally, we should test the embeddings on uncleaned data or more challenging cases to further validate their robustness. Beyond static embeddings, future experiments could explore the impact of other parameters, such as learning rate and image size, to optimize performance. These insights will influence how we design experiments in the future, focusing on embedding strategies that encode domain-specific knowledge.

Contributions

Experiments by Beamlak Bekel date range 7/16/2024 -8/21/2024