MinishLab / **model2vec** Public

The Fastest State-of-the-Art Static Embeddings in the World

minishlab.github.io/

MIT license

999 stars 44 forks Branches Tags Activity

Star Notifications

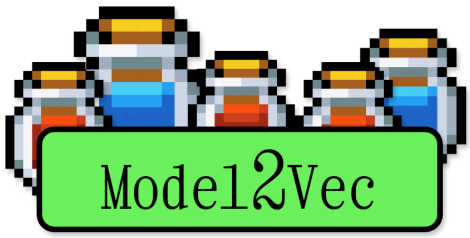
Code Issues 1 Pull requests 1 Discussions Actions Projects Security Insights

main 5 Branches 17 Tags Go to file Go to file Code

Pringled docs: Update plot (#169) 44b0dcf · 3 days ago

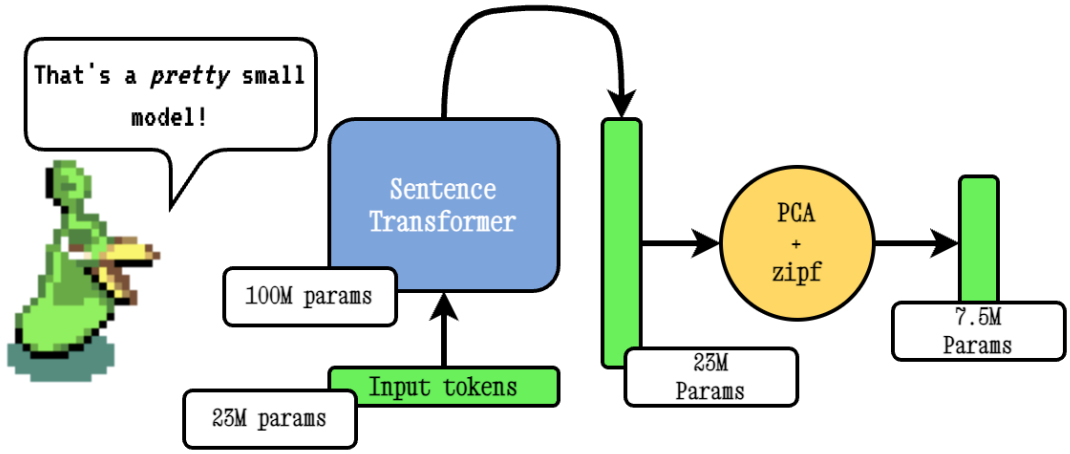
.github/workflows	fix: Fixed CI (#124)	3 months ago
assets/images	docs: Update plot (#169)	3 days ago
model2vec	increase version (#166)	last week
results	docs: Added new model results (#167)	4 days ago
scripts	feat: Added onnx and tokenizer files support scr...	4 months ago
tests	feat: float pca dims (#163)	last week
tutorials	remove deduplication tutorial (#159)	last week
.gitignore	docs: Move results and add blogpost (#82)	4 months ago
.pre-commit-config.yaml	tests: Add distill tests and CI (#42)	5 months ago
LICENSE	Initial commit	7 months ago
Makefile	feat: add support for pattern for unused tokens....	2 months ago
README.md	docs: Added new model results (#167)	4 days ago
pyproject.toml	feat: Add multiprocessing (#141)	2 months ago
uv.lock	feat: Updated save_pretrained to save sentence-...	2 weeks ago

README MIT license



The Fastest State-of-the-Art Static Embeddings in the World

🤗 Models | 📚 Tutorials | 🌐 Website | 🏆 Results



Model2Vec is a technique to turn any sentence transformer into a really small static model, reducing model size by 15x and making the models up to 500x faster, with a small drop in performance. Our best model is the most performant static embedding model in the world. See our results here, or dive in to see how it works.

Updates & Announcements

- **30/01/2024**: We released two new models: potion-base-32M and potion-retrieval-32M. potion-base-32M is our most performant model to date, using a larger vocabulary and higher dimensions. potion-retrieval-32M is a finetune of potion-base-32M that is optimized for retrieval tasks, and is the best performing static retrieval model currently available.
- **30/10/2024**: We released three new models: potion-base-8M, potion-base-4M, and potion-base-2M. These models are trained using Tokenlearn. Find out more in our blog post. NOTE: for users of any of our old English M2V models, we recommend switching to these new models as they perform better on all tasks.

Table of Contents

Quickstart

Install the package with:

```
pip install model2vec
```

This will install the base inference package, which only depends on `numpy` and a few other minor dependencies. If you want to distill your own models, you can install the distillation extras with:

```
pip install model2vec[distill]
```

The easiest way to get started with Model2Vec is to load one of our flagship models from the HuggingFace hub. These models are pre-trained and ready to use. The following code snippet shows how to load a model and make embeddings:

```python
from model2vec import StaticModel

# Load a model from the HuggingFace hub (in this case the potion-base-8M model)
model = StaticModel.from_pretrained("minishlab/potion-base-8M")

# Make embeddings
embeddings = model.encode(["It's dangerous to go alone!", "It's a secret to everybody."])

# Make sequences of token embeddings
token_embeddings = model.encode_as_sequence(["It's dangerous to go alone!", "It's a secret to everybody."])
```

And that's it. You can use the model to classify texts, to cluster, or to build a RAG system.

Instead of using one of our models, you can also distill your own Model2Vec model from a Sentence Transformer model. The following code snippet shows how to distill a model:

```python
from model2vec.distill import distill

# Distill a Sentence Transformer model, in this case the BAAI/bge-base-en-v1.5 model
m2v_model = distill(model_name="BAAI/bge-base-en-v1.5", pca_dims=256)

# Save the model
m2v_model.save_pretrained("m2v_model")
```

Distillation is really fast and only takes 30 seconds on CPU. Best of all, distillation requires no training data.

For advanced usage, such as using Model2Vec in the Sentence Transformers library, please refer to the Usage sections.

Main Features

- **State-of-the-Art Performance**: Model2Vec models outperform any other static embeddings (such as GLoVe and BPEmb) by a large margin, as can be seen in our results.
- **Small**: Model2Vec reduces the size of a Sentence Transformer model by a factor of 15, from 120M params, down to 7.5M (30 MB on disk, making it the smallest model on MTEB!).
- **Lightweight Dependencies**: the base package's only major dependency is `numpy`.
- **Lightning-fast Inference**: up to 500 times faster on CPU than the original model. Go green or go home.
- **Fast, Dataset-free Distillation**: distill your own model in 30 seconds on a CPU, without a dataset. All you need is a model and (optionally) a custom vocabulary.
- **Integrated in many popular libraries**: Model2Vec can be used directly in popular libraries such as Sentence Transformers, LangChain, txtai, and Chonkie. See the Integrations section for more information.
- **Tightly integrated with HuggingFace hub**: easily share and load models from the HuggingFace hub, using the familiar `from_pretrained` and `push_to_hub`. Our own models can be found here. Feel free to share your own.

What is Model2Vec?

Model2vec creates a small, fast, and powerful model that outperforms other static embedding models by a large margin on all tasks we could find, while being much faster to create than traditional static embedding models such as GloVe. Like BPEmb, it can create subword embeddings, but with much better performance. Distillation doesn't need *any* data, just a vocabulary and a model.

The base model2vec technique works by passing a vocabulary through a sentence transformer model, then reducing the dimensionality of the resulting embeddings using PCA, and finally weighting the embeddings using zipf weighting. During inference, we simply take the mean of all token embeddings occurring in a sentence.

Our potion models are pre-trained using tokenlearn, a technique to pre-train model2vec distillation models. These models are created with the following steps:

- **Distillation**: We distill a Model2Vec model from a Sentence Transformer model, using the method described above.
- **Sentence Transformer inference**: We use the Sentence Transformer model to create mean embeddings for a large number of texts from a corpus.
- **Training**: We train a model to minimize the cosine distance between the mean embeddings generated by the Sentence Transformer model and the mean embeddings generated by the Model2Vec model.
- **Post-training re-regularization**: We re-regularize the trained embeddings by first performing PCA, and then weighting the embeddings using `smooth inverse frequency (SIF)` weighting using the following formula: `w = 1e-3 / (1e-3 + proba)`. Here, `proba` is the probability

of the token in the corpus we used for training.

For a much more extensive deepdive, please refer to our Model2Vec blog post and our Tokenlearn blog post.

Usage

Inference

▶ Inference with a pretrained model

▶ Inference with the Sentence Transformers library

Distillation

▼ Distilling from a Sentence Transformer

The following code can be used to distill a model from a Sentence Transformer. As mentioned above, this leads to really small model that might be less performant.

```python
from model2vec.distill import distill

# Distill a Sentence Transformer model
m2v_model = distill(model_name="BAAI/bge-base-en-v1.5", pca_dims=256)

# Save the model
m2v_model.save_pretrained("m2v_model")
```

▼ Distilling from a loaded model

If you already have a model loaded, or need to load a model in some special way, we also offer an interface to distill models in memory.

```python
from transformers import AutoModel, AutoTokenizer

from model2vec.distill import distill_from_model

# Assuming a loaded model and tokenizer
model_name = "baai/bge-base-en-v1.5"
model = AutoModel.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)

m2v_model = distill_from_model(model=model, tokenizer=tokenizer, pca_dims=256)

m2v_model.save_pretrained("m2v_model")
```

▼ Distilling with the Sentence Transformers library

The following code snippet shows how to distill a model using the Sentence Transformers library. This is useful if you want to use the model in a Sentence Transformers pipeline.

```python
from sentence_transformers import SentenceTransformer
from sentence_transformers.models import StaticEmbedding

static_embedding = StaticEmbedding.from_distillation("BAAI/bge-base-en-v1.5", device="cpu", pca_dims=256)
model = SentenceTransformer(modules=[static_embedding])
embeddings = model.encode(["It's dangerous to go alone!", "It's a secret to everybody."])
```

▼ Distilling with a custom vocabulary

If you pass a vocabulary, you get a set of static word embeddings, together with a custom tokenizer for exactly that vocabulary. This is comparable to how you would use GLoVe or traditional word2vec, but doesn't actually require a corpus or data.

```python
from model2vec.distill import distill

# Load a vocabulary as a list of strings
vocabulary = ["word1", "word2", "word3"]

# Distill a Sentence Transformer model with the custom vocabulary
m2v_model = distill(model_name="BAAI/bge-base-en-v1.5", vocabulary=vocabulary)
```

```
# Save the model
m2v_model.save_pretrained("m2v_model")

# Or push it to the hub
m2v_model.push_to_hub("my_organization/my_model", token="<it's a secret to everybody>")
```

By default, this will distill a model with a subword tokenizer, combining the models (subword) vocab with the new vocabulary. If you want to get a word-level tokenizer instead (with only the passed vocabulary), the `use_subword` parameter can be set to `False`, e.g.:

```
m2v_model = distill(model_name=model_name, vocabulary=vocabulary, use_subword=False)
```

Important note: we assume the passed vocabulary is sorted in rank frequency. i.e., we don't care about the actual word frequencies, but do assume that the most frequent word is first, and the least frequent word is last. If you're not sure whether this is case, set `apply_zipf` to `False`. This disables the weighting, but will also make performance a little bit worse.

Evaluation

▼ Installation

Our models can be evaluated using our evaluation package. Install the evaluation package with:

```
pip install git+https://github.com/MinishLab/evaluation.git@main
```

▼ Evaluation Code

The following code snippet shows how to evaluate a Model2Vec model:

```
from model2vec import StaticModel

from evaluation import CustomMTEB, get_tasks, parse_mteb_results, make_leaderboard, summarize_results
from mteb import ModelMeta

# Get all available tasks
tasks = get_tasks()
# Define the CustomMTEB object with the specified tasks
evaluation = CustomMTEB(tasks=tasks)

# Load the model
model_name = "m2v_model"
model = StaticModel.from_pretrained(model_name)

# Optionally, add model metadata in MTEB format
model.mteb_model_meta = ModelMeta(
            name=model_name, revision="no_revision_available", release_date=None, languages=None
        )

# Run the evaluation
results = evaluation.run(model, eval_splits=["test"], output_folder=f"results")

# Parse the results and summarize them
parsed_results = parse_mteb_results(mteb_results=results, model_name=model_name)
task_scores = summarize_results(parsed_results)

# Print the results in a leaderboard format
print(make_leaderboard(task_scores))
```

Integrations

▼ Sentence Transformers

Model2Vec can be used directly in Sentence Transformers using the `StaticEmbedding` module.

The following code snippet shows how to load a Model2Vec model into a Sentence Transformer model:

```
from sentence_transformers import SentenceTransformer
from sentence_transformers.models import StaticEmbedding

# Initialize a StaticEmbedding module
static_embedding = StaticEmbedding.from_model2vec("minishlab/potion-base-8M")
```

```
model = SentenceTransformer(modules=[static_embedding])
embeddings = model.encode(["It's dangerous to go alone!", "It's a secret to everybody."])
```

The following code snippet shows how to distill a model directly into a Sentence Transformer model:

```
from sentence_transformers import SentenceTransformer
from sentence_transformers.models import StaticEmbedding

static_embedding = StaticEmbedding.from_distillation("BAAI/bge-base-en-v1.5", device="cpu", pca_dims=256)
model = SentenceTransformer(modules=[static_embedding])
embeddings = model.encode(["It's dangerous to go alone!", "It's a secret to everybody."])
```

For more documentation, please refer to the Sentence Transformers documentation.

▼ LangChain

Model2Vec can be used in LangChain using the `langchain-community` package. For more information, see the LangChain Model2Vec docs. The following code snippet shows how to use Model2Vec in LangChain after installing the `langchain-community` package with `pip install langchain-community`:

```
from langchain_community.embeddings import Model2vecEmbeddings
from langchain_community.vectorstores import FAISS
from langchain.schema import Document

# Initialize a Model2Vec embedder
embedder = Model2vecEmbeddings("minishlab/potion-base-8M")

# Create some example texts
texts = [
    "Enduring Stew",
    "Hearty Elixir",
    "Mighty Mushroom Risotto",
    "Spicy Meat Skewer",
    "Fruit Salad",
]

# Embed the texts
embeddings = embedder.embed_documents(texts)

# Or, create a vector store and query it
documents = [Document(page_content=text) for text in texts]
vector_store = FAISS.from_documents(documents, embedder)
query = "Risotto"
query_vector = embedder.embed_query(query)
retrieved_docs = vector_store.similarity_search_by_vector(query_vector, k=1)
```

▼ Txtai

Model2Vec can be used in txtai for text embeddings, nearest-neighbors search, and any of the other functionalities that txtai offers. The following code snippet shows how to use Model2Vec in txtai after installing the `txtai` package (including the `vectors` dependency) with `pip install txtai[vectors]`:

```
from txtai import Embeddings

# Load a model2vec model
embeddings = Embeddings(path="minishlab/potion-base-8M", method="model2vec", backend="numpy")

# Create some example texts
texts = ["Enduring Stew", "Hearty Elixir", "Mighty Mushroom Risotto", "Spicy Meat Skewer", "Chilly Fruit Salad"]

# Create embeddings for downstream tasks
vectors = embeddings.batchtransform(texts)

# Or create a nearest-neighbors index and search it
embeddings.index(texts)
result = embeddings.search("Risotto", 1)
```

▼ Chonkie

Model2Vec is the default model for semantic chunking in Chonkie. To use Model2Vec for semantic chunking in Chonkie, simply install Chonkie with `pip install chonkie[semantic]` and use one of the `potion` models in the `SemanticChunker` class. The following code snippet shows how to use Model2Vec in Chonkie:

```python
from chonkie import SDPMChunker

# Create some example text to chunk
text = "It's dangerous to go alone! Take this."

# Initialize the SemanticChunker with a potion model
chunker = SDPMChunker(
    embedding_model="minishlab/potion-base-8M",
    similarity_threshold=0.3
)

# Chunk the text
chunks = chunker.chunk(text)
```

▼ Transformers.js

To use a Model2Vec model in transformers.js, the following code snippet can be used as a starting point:

```javascript
import { AutoModel, AutoTokenizer, Tensor } from '@huggingface/transformers';

const modelName = 'minishlab/potion-base-8M';

const modelConfig = {
    config: { model_type: 'model2vec' },
    dtype: 'fp32',
    revision: 'refs/pr/1'
};
const tokenizerConfig = {
    revision: 'refs/pr/2'
};

const model = await AutoModel.from_pretrained(modelName, modelConfig);
const tokenizer = await AutoTokenizer.from_pretrained(modelName, tokenizerConfig);

const texts = ['hello', 'hello world'];
const { input_ids } = await tokenizer(texts, { add_special_tokens: false, return_tensor: false });

const cumsum = arr => arr.reduce((acc, num, i) => [...acc, num + (acc[i - 1] || 0)], []);
const offsets = [0, ...cumsum(input_ids.slice(0, -1).map(x => x.length))];

const flattened_input_ids = input_ids.flat();
const modelInputs = {
    input_ids: new Tensor('int64', flattened_input_ids, [flattened_input_ids.length]),
    offsets: new Tensor('int64', offsets, [offsets.length])
};

const { embeddings } = await model(modelInputs);
console.log(embeddings.tolist()); // output matches python version
```

Note that this requires that the Model2Vec has a `model.onnx` file and several required tokenizers file. To generate these for a model that does not have them yet, the following code snippet can be used:

```bash
python scripts/export_to_onnx.py --model_path <path-to-a-model2vec-model> --save_path "<path-to-save-the-onnx-model>"
```

Model List

We provide a number of models that can be used out of the box. These models are available on the HuggingFace hub and can be loaded using the `from_pretrained` method. The models are listed below.

Model	Language	Vocab	Sentence Transformer	Tokenizer Type	Params	Tokenlearn
potion-base-32M	English	Output + Frequent C4 tokens	bge-base-en-v1.5	Subword	32.3M	✅

Model	Language	Vocab	Sentence Transformer	Tokenizer Type	Params	Tokenlearn
potion-base-8M	English	Output	bge-base-en-v1.5	Subword	7.5M	✅
potion-base-4M	English	Output	bge-base-en-v1.5	Subword	3.7M	✅
potion-base-2M	English	Output	bge-base-en-v1.5	Subword	1.8M	✅
potion-retrieval-32M	English	Output + Frequent C4 tokens	bge-base-en-v1.5	Subword	32.3M	✅
M2V_multilingual_output	Multilingual	Output	LaBSE	Subword	471M	❌

Results

We have performed extensive experiments to evaluate the performance of Model2Vec models. The results are documented in the results folder

Releases 17

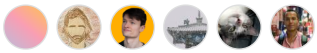
v0.3.8 Latest
last week

+ 16 releases

Packages

No packages published

Contributors 6



Languages

Python 76.2% Jupyter Notebook 23.6% Makefile 0.2%