modelstore

Neal Lathia

Jan 30, 2021

CONTENTS:

`modelstore` is a Python library that allows you to version, export, and save/retrieve machine learning models to and from your filesystem or a cloud storage provider (AWS or GCP).

The library's `ModelStore` automates versioning your models, storing them in a structured way, retrieving them, and collecting meta data about the Python runtime that was used to train them.

CHAPTER

ONE

INSTALLING THE MODELSTORE LIBRARY

This library can be installed via pip:

```
pip install modelstore
```

You can find the latest version here: modelstore on Pypi.

CHAPTER

TWO

QUICK START

This library's `ModelStore` enables you to export trained ML models and store it to your choice of storage.

2.1 Create a model store instance

`modelstore` currently supports storing models to:

- A directory in a local file system
- Google Cloud buckets: set up a Google Cloud project and have create a cloud storage bucket.
- AWS S3 buckets: set up a project and create an s3 bucket.
- A storage service that we manage for you. This requires you to have API keys.

To save your models, create a model store instance with one of the following:

```
from modelstore import ModelStore

# A local file system
model_store = ModelStore.from_file_system(
    root="/path/to/directory",
)

# Google cloud bucket
model_store = ModelStore.from_gcloud(
    project_name="my-project",
    bucket_name="my-bucket",
)

# AWS S3 bucket
model_store = ModelStore.from_aws_s3(
    bucket_name="my-bucket",
)

# A managed storage service
model_store = ModelStore.from_api_key(
    access_key_id="<your-access-key-id>",
    secret_access_key="<your-secret-access-key>"
)
```

2.2 Upload a model to the model store

The `modelstore` library has separate up functions to store models that were trained with different ML libraries, such as scikit-learn or tensorflow. They all follow the same pattern.

For example, to store a `scikit-learn` model, use:

```
model_store.sklearn.upload(domain="domain-name", model=my_model)
```

When you upload a model, you need to specify a **domain**. This is the string that groups several models that are for the same end-usage together. For example, let's assume you are training several models to predict whether an email is spam. Setting `domain="spam-detection"` will store all of those models together, and you will then be able to list and retrieve them all.

To read more about the supported libraries, see: *Supported Machine Learning Libraries*.

To read more about how this library organises models, see *The Model Store Structure*.

2.3 Download a model from the model store

To retrieve a model from your chosen storage, use `download()`:

```
file_path = model_store.download(
   local_path=".", # Where to download the model to
   domain="example-model", # The model's domain
   model_id="model-id"  # Optional; the ID of the specific model
)
```

If you do not provide a `model_id` parameter, the `download()` function will default to the last model that was stored for the given domain.

CHAPTER

THREE

THE MODEL STORE STRUCTURE

This library's model store interacts with a backend of your chosing. The library currently supports:

- A local file store
- Google Cloud Storage
- AWS S3 Buckets

If you do not want to manage your own storage system, we also have a hosted storage that you can use with an API key.

This library stores models in cloud buckets using a pre-defined structure.

3.1 Model Archive & Meta Data

When you use `upload()`, an `artifacts.tar.gz` file is created and then uploaded to the storage of your choice. This archive contains:

1. Any files that were dumped from your model,
2. A `"python-info.json"` file that enumerates the version of the Python library of the model you are exporting.

The `upload()` function returns a dictionary containing meta-data about the model.

The meta-data includes:

- A unique *UUID4* for your model;
- Details about where the model is being uploaded to (the bucket and prefix);
- The Python runtime that was used (e.g., "python:3.7.0")
- The user who ran the training.
- Versions for the Python library and key dependencies.

3.2 Model Domains

A **domain** is the word we use to group models, that are all intended for the same end-usage, together.

Under the hood, this is just a string, so it is up to you how you would like to use it; it is required because this library stores models by domain.

3.3 File Storage Structure

When you pick a backend that stores data in files (e.g., Cloud Storage Buckets), the files are stored with a pre-defined structure.

The top-level, **root** prefix that this library hard-codes is `operatorai-model-store`.

When you create and upload a model archive, this library will upload three files to different places in the bucket.

1. **The artifacts archive** will be uploaded to: `root/<domain>/<datetime>/archive.tar.gz`, where the datetime has the form `"%Y/%m/%d/%H:%M:%S"` - denoting the time when the model was uploaded. 2. The library creates a dictionary of **meta-data** about your model. This will be uploaded to `root/<domain>/versions/<model-id>.json`. 3. This same meta-data is also stored in `root/<domain>/latest.json`, which tracks the _last_ model that was uploaded to the model store.

3.3.1 Example

Let's imagine you're training a text classifier to detect whether some customer text is about "refunds."

Over time, you may end up re-training this classifier several times, with newer data or different models types; however, you still need a way to denote that all of these models were about detecting refund requests.

In this case, you could set the `domain="customer-refunds"`.

Models that are exported in this domain will be stored to:

```
<root>/<domain>/<time/of/upload>/artifacts.tar.gz
operatorai-model-store/customer-refunds/2020/08/30/23:29:28/artifacts.tar.gz
```

CHAPTER

FOUR

SUPPORTED MACHINE LEARNING LIBRARIES

This library currently supports:

- CatBoost
- Keras
- LightGBM
- PyTorch
- PyTorch Lightning
- Scikit Learn
- Tensorflow
- Transformers
- XGBoost

The common pattern, across all supported libraries, is to:

```
# Create an instance of the model store
from modelstore import ModelStore

model_store = ModelStore.from_gcloud(
   project_name="my-project",
   bucket_name="my-bucket",
)

# Upload your model by calling `upload()`
model_store.<library-name>.upload("my-domain", ...)
```

4.1 CatBoost

To export a CatBoost model, use:

```
# Train your model
model = ctb.CatBoostClassifier(loss_function="MultiClass")
model.fit(x, y)

# Upload the model
model_store.catboost.upload("my-domain", model=clf, pool=df)
```

This will store a multiple formats of your model to the model store:

- CatBoost binary format
- JSON
- Onnx

The `pool` argument is required if you are training a multi class model. The stored model will also contain a `model_attributes.json` file with all of the attributes of the model.

4.2 Keras

To export a Keras model, use:

```
# Train your model
model = keras.Model(inputs, outputs)
model.compile(optimizer="adam", loss="mean_squared_error")
model.fit(X_train, y_train, epochs=10)
# ...

# Upload the model
model_store.keras.upload("my-domain", model=net, optimizer=optim)
```

This will create two dumps of the model, based on calling `model.to_json()` and `model.save()`.

4.3 LightGBM

To export a LightGBM model, use:

```
# Train your model
model = lgb.train(param, train_data, num_round, valid_sets=[validation_data])
# ...

# Upload the model
model_store.lightgbm.upload(model_domain, model=model)
```

This will create two dumps of the model, based on calling `model.save_model()` and `model.dump_model()`.

4.4 PyTorch

To export a PyTorch model, use:

```
# Train your model
net = ExampleNet()
optim = ExampleOptim()
# ...

# Upload the model
model_store.pytorch.upload("my-domain", model=net, optimizer=optim)
```

This will create two dumps of the model; a `checkpoint.pt` that contains the net and optimizer's state (e.g., to continue training at a later date), and a `model.pt` that is the result of `torch.save` with the model only (e.g., for inference).

4.5 PyTorch Lightning

To export a PyTorch Lightning model, use:

```
# Train your model
model = ExampleLightningNet()
trainer = pl.Trainer(max_epochs=5, default_root_dir=mkdtemp())
trainer.fit(model, train_dataloader, val_dataloader)

# Upload the model
model_store.pytorch_lightning.upload(
    model_domain, trainer=trainer, model=model
)
```

This will create a dump of the model; based on calling the `trainer.save_checkpoint(file_path)` function.

4.6 Scikit-Learn

To export a scikit-learn model, use:

```
# Train your model
clf = RandomForestClassifier(n_estimators=10)
clf = clf.fit(X, Y)

# Upload the model
model_store.sklearn.upload("my-domain", model=clf)
```

This will create a `joblib` dump of the model.

4.7 Tensorflow

To export a tensorflow model, use:

```
# Train your model
model = tf.keras.models.Sequential(
    [
        tf.keras.layers.Dense(5, activation="relu", input_shape=(10,)),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(1),
    ]
)
model.compile(optimizer="adam", loss="mean_squared_error")
model.fit(X_train, y_train, epochs=10)

# Upload the model
model_store.tensorflow.upload("my-domain", model=model)
```

This will both save the weights (as a checkpoint file) and export/save the entire model.

4.8 Transformers

To export a transformers model, use:

```
# Get a pre-trained model and fine tune it
model_name = "distilbert-base-cased"
config = AutoConfig.from_pretrained(
    model_name, num_labels=2, finetuning_task="mnli",
)
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(
    model_name, config=config,
)

# Upload the model
model_store.transformers.upload(
    "my-domain", config=config, model=model, tokenizer=tokenizer,
)
```

The `config` and `tokenizer` parameters are optional. This will use the `save_pretrained()` function to save your model.

4.9 XGBoost

To export an XGBoost model, use:

```
# Train your model
bst = xgb.train(param, dtrain, num_round)

# Upload the model
model_store.xgboost.upload("my-domain", model=bst)
```

This will add two dumps of the model into the archive; a model dump (in an interchangeable format, for loading again later), and a model save (in JSON format, which, to date, is experimental).

CHAPTER

FIVE

SCIKIT-LEARN EXAMPLE

This example is based on the GradientBoostingRegressor tutorial from the scikit-learn website:

```
import json
import os

from sklearn.datasets import load_diabetes
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split

from modelstore import ModelStore


def train():
    diabetes = load_diabetes()
    X_train, X_test, y_train, y_test = train_test_split(
        diabetes.data, diabetes.target, test_size=0.1, random_state=13
    )
    params = {
        "n_estimators": 500,
        "max_depth": 4,
        "min_samples_split": 5,
        "learning_rate": 0.01,
        "loss": "ls",
    }
    reg = GradientBoostingRegressor(**params)
    reg.fit(X_train, y_train)
    # Skipped for brevity (but important!) evaluate the model
    return reg


if __name__ == "__main__":
    # In this demo, we train a GradientBoostingRegressor
    # using the same approach described on the scikit-learn website.
    # Replace this with the code to train your own model
    model = train()

    # The modelstore library currently assumes you have already created
    # a Cloud Storage bucket and will raise an exception if it doesn't exist

    # This example assumes that you have the GCP project name and bucket id
    # saved as environment variables - replace the os.environ below with
    # your values
    store = ModelStore.from_gcloud(
        project_name=os.environ["GCP_PROJECT_ID"],
```

(continues on next page)

(continued from previous page)

```
        bucket_name=os.environ["GCP_BUCKET_NAME"],
    )

    # Upload the model
    meta_data = store.sklearn.upload(
        "sklearn-diabetes-boosting-demo",
        model=model
    )

    # The upload returns meta-data about the model that was uploaded
    # This meta-data has also been sync'ed into the cloud storage
    #  bucket
    print("  Finished uploading model!")
    print(json.dumps(meta_data, indent=4))

    # Download the model back!
    target = f"downloaded-{model_type}-model"
    os.makedirs(target, exist_ok=True)
    model_path = model_store.download(
        local_path=target,
        domain=model_domain,
        model_id=meta["model"]["model_id"],
    )
    print(f"  Downloaded the model back to {model_path}")
```

CHAPTER

SIX

LICENSE

CHAPTER

SEVEN

CONTACT

If you have any questions or feedback, feel free to email me: `neal.lathia@gmail.com`.

If you want to follow along as this (and other) tools are developed, sign up here.