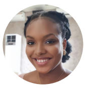
Tutorials March 8, 2023 *7 min read*

How to evaluate and load a PyTorch model with Giskard?

This tutorial teaches you how to upload a PyTorch model (built from scratch or pre-trained) to Giskard, and identify potential errors and biases.


Favour Kelvin

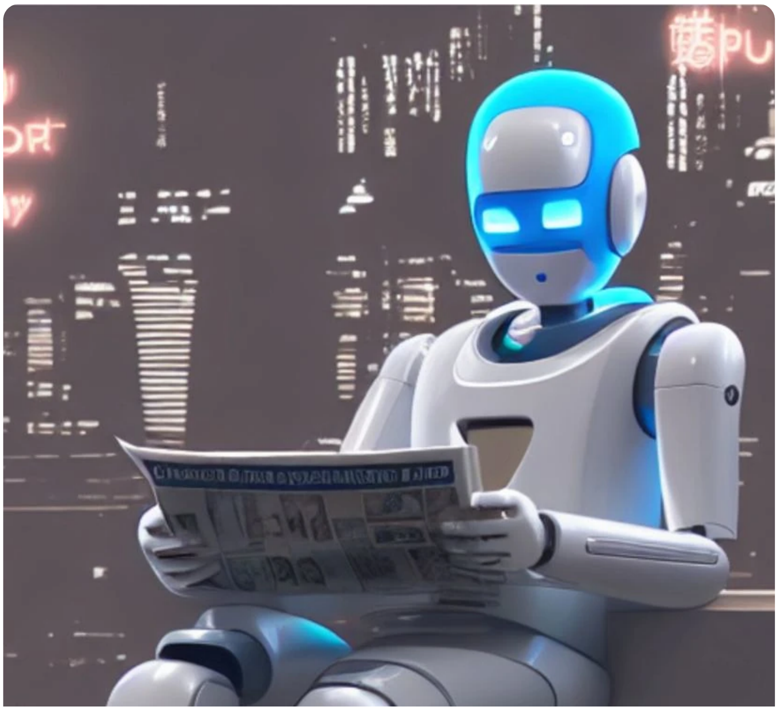


Table of contents

Note: The API used by this tutorial is deprecated. To use the new API, please refer to the official documentation: https://docs.giskard.ai

🔥 Training and testing a PyTorch model

Have you ever spent hours training a PyTorch model, only to realize you made a mistake? Or have you ever gotten decent results from your model, but you are not sure if it is because you built the model correctly or just because deep learning is so powerful that even a flawed architecture can produce acceptable outcomes? AI is not perfect. This trade-off is something to be aware of, especially when considering the limitations of AI and its suitability for various problems. However, AI has previously considered almost impossible.

Training a PyTorch model can be a time-consuming process, so it is frustrating when you realize you made a mistake in the code. It is also hard to know if your model is truly effective, or if it is just luck that it's producing good results. To ensure your model will work well on new data, it is important to test it and compare the accuracy and loss to the values you saw at the end of training. If there's a significant difference, it could mean the model is overfitted to the training data and may not perform well on unseen data.

Giskard is a tool designed to address some of the challenges of working with AI. It allows you to quickly test your model to make sure there are no biases and errors in your model. In this tutorial, we will guide you through how tuned pretrained model to Giskard for analysis to find edge cases and bugs.

⏪ Before loading your PyTorch model

You'll need to have Git and Docker installed. To get started, follow these installation instructions to install the Giskard Python library.

🏃Build a PyTorch model and analyse it with Giskard

If you have a model which was built from scratch, there is a chance that it contains performance issues and edge cases. Giskard helps to detect problems related to model performance, robustness, discriminative behaviour, etc.

Below are the steps to build, scan and upload a PyTorch model, which based

```
import time

import torch
import numpy as np
import pandas as pd
from torch import nn
from torchtext.datasets import AG_NEWS
from torch.utils.data import DataLoader
from sklearn.metrics import accuracy_score
from torchtext.data.utils import
get_tokenizer
from torch.utils.data.dataset import
random_split
from torchtext.vocab import
build_vocab_from_iterator
from torchtext.data.functional import
to_map_style_dataset


# Define constants.
DEVICE = torch.device("cpu")

TARGET_MAP = {0: "World", 1: "Sports", 2:
"Business", 3: "Sci/Tech"}
TARGET_COLUMN_NAME = "label"
FEATURE_COLUMN_NAME = "text"

LOADERS_BATCH_SIZE = 64


# Fetch data.
train_data, test_data = AG_NEWS()

# Simple English tokenizer provided by
torchtext.
tokenizer = get_tokenizer("basic_english")

# Build a vocabulary from all the tokens we
can find in the train data.
vocab =
build_vocab_from_iterator((tokenizer(text)
```

```
    return vocab(tokenizer(raw_text))


def preprocess_label(raw_label):
    return int(raw_label) - 1


def collate_fn(batch):
    label_list, text_list, offsets = [], [], [0]

    for _label, _text in batch:
        label_list.append(preprocess_label(_label))
        processed_text = torch.tensor(preprocess_text(_text), dtype=torch.int64)
        text_list.append(processed_text)
        offsets.append(processed_text.size(0))

    label_list = torch.tensor(label_list, dtype=torch.int64)
    offsets = torch.tensor(offsets[:-1]).cumsum(dim=0)
    text_list = torch.cat(text_list)

    return label_list.to(DEVICE), text_list.to(DEVICE), offsets.to(DEVICE)


# Create the datasets.
train_dataset = to_map_style_dataset(train_data)
test_dataset = to_map_style_dataset(test_data)

# We further divide the training data into a train and validation split.
train_split, valid_split = random_split(train_dataset, [0.95, 0.05])

```

```python
test_dataloader  = DataLoader(test_dataset, batch_size=LOADERS_BATCH_SIZE, shuffle=True, collate_fn=collate_fn)
```


2. Wrap a dataset with Giskard

We need to wrap a dataset with giskard.Dataset. This wrapper allows to perform model scanning on a given data.

```python
from giskard import Dataset


# Prepare data to wrap.
raw_data = pd.DataFrame({TARGET_COLUMN_NAME: TARGET_MAP[label_id - 1], FEATURE_COLUMN_NAME: text} for label_id, text in test_data)

# Wrap it with Giskard.
wrapped_data = Dataset(raw_data, name="Test Dataset", target="label", column_types={FEATURE_COLUMN_NAME: "text", TARGET_COLUMN_NAME: "category"})
```


3. Build your PyTorch model

```python
# Define model.
class TextClassificationModel(nn.Module):
```

```
        self.fc = nn.Linear(embed_dim, num_class)
        self.init_weights()

    def init_weights(self):
        init_range = 0.5
        self.embedding.weight.data.uniform_(-init_range, init_range)
        self.fc.weight.data.uniform_(-init_range, init_range)
        self.fc.bias.data.zero_()

    def forward(self, text, offsets):
        embedded = self.embedding(text, offsets)
        return self.fc(embedded).softmax(axis=-1)


model = TextClassificationModel(vocab_size=len(vocab), embed_dim=64, num_class=4).to(DEVICE)


# Train and evaluate model.
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=5)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, 1, gamma=0.1)


def train_epoch(dataloader):
    model.train()

    train_accuracy = total_count = 0
    for label, text, offset in dataloader:
        optimizer.zero_grad()
        predicted_label = model(text, offset)
        loss = criterion(predicted_label, label)
```

```
(predicted_label.argmax(1) ==
label).sum().item()
            total_count += label.size(0)

    return train_accuracy / total_count


def validation_epoch(dataloader):
    model.eval()

    validation_accuracy = total_count = 0
    with torch.no_grad():
        for label, text, offsets in
dataloader:
            predicted_label = model(text,
offsets)
            validation_accuracy +=
(predicted_label.argmax(1) ==
label).sum().item()
            total_count += label.size(0)

    return validation_accuracy / total_count


total_accuracy = None
for epoch in range(1, 3):
    start_time = time.perf_counter()

    train_epoch(train_dataloader)
    accu_val =
validation_epoch(valid_dataloader)

    if total_accuracy is not None and
total_accuracy > accu_val:
        scheduler.step()
    else:
        total_accuracy = accu_val

    print("-" * 65)
    print(f"| end of epoch {epoch: .3f} |
time: {time.perf_counter() - start_time
:5.2f}s | valid accuracy {accu_val:8.3f} ")
```

```
print(f"Test accuracy {test_accuracy:.3f}")
```


4. Define a prediction function

In order to encapsulate the prediction logic, we need to put it inside a "prediction_function", which will be used to infer the predictions during the analysis with Giskard.

```
def infer_predictions(_model: torch.nn.Module, _dataloader: DataLoader) -> np.ndarray:
    _model.eval()
    pred = list()

    for _, text, offsets in _dataloader:
        with torch.no_grad():
            probs = model(text, offsets).cpu().detach().numpy()

        pred.append(probs)

    pred = np.concatenate(pred, axis=0)
    return pred


def prediction_function(df: pd.DataFrame) -> np.ndarray:
    # Placeholder for label.
    if df.shape[1] == 1:
        df.insert(0, TARGET_COLUMN_NAME, np.zeros(len(df)))

```

```python
                                          collate_fn=collate_fn)
    predictions = infer_predictions(model, dataloader)
    return predictions
```


5. Wrap your model with Giskard

Same as with wrapping the dataset, we need to wrap the model to perform an analysis using the Giskard framework. This is done via the giskard.Model wrapper.

```python
from giskard import Model


# Wrap model with Giskard.
wrapped_model = Model(model=prediction_function,
                      name="SimpleNewsClassificationModel",
                      feature_names=["text"],
                      model_type="classification",
                      classification_labels=list(TARGET_MAP.values()))

# Validate wrapped model.
wrapped_test_metric = accuracy_score(wrapped_data.df[TARGET_COLUMN_NAME],
```

6. Scan your model with Giskard

Now that we are all set to analyse trained model, we will use giskard.scan to find problematic issues.

```python
from giskard import scan


results = scan(wrapped_model, wrapped_data)
display(results)
```

giskard_twitter_sentiment_05_model_scanning.py view raw
hosted with ❤️ by GitHub

By calling display(results) one can conveniently view detected issues directly in the jupyter notebook. For example, the given model is not robust to typos in the text.

7. Generate tests based on the scanning results

the changes he applied in the new version of a model helped to resolve the detected problems:

```python
test_suite = results.generate_test_suite("My first test suite")
test_suite.run()
```

view raw
giskard_twitter_sentiment_06_test_suites_generation.py hosted with ❤️ by GitHub

By calling test_suite.run(), the user can also visualise the result of test-suite execution.

🔍 Further analysis with Giskard UI

Additionally to the Giskard Python library, the Giskard server is the app that you can install either locally or on your cloud instance. It provides a convenient UI to debug tests, compare models, collaborate with other users, etc. To learn more about its functionality check the documentation.

```
giskard server start
```

giskard_general_server_start hosted with ❤ by GitHub view raw

2. Start the Giskard worker

Next, we need to setup the giskard worker. This is the machine, which executes test-suites and dataset inspection.

```
giskard worker start -u
http://localhost:19000/
```

giskard_general_worker_start hosted with ❤ by GitHub view raw

3. Upload your model and dataset to the Giskard server

Now, we are all set to upload the necessary artefacts, including dataset, trained model and generated test suite to the Giskard UI:

```
from giskard import GiskardClient


# Uploading the test suite will automatically
save the model, dataset, tests, slicing &
```

```python

client = GiskardClient(
    url="http://localhost:19000",  # URL of your Giskard instance
    token=token
)

my_project = client.create_project("project_id", "PROJECT_NAME", "Project description.")

# Upload to the current project ✉
test_suite.upload(client, "project_id")
```

giskard_twitter_sentiment_07_artifacts_upload.py view raw
hosted with ❤ by GitHub

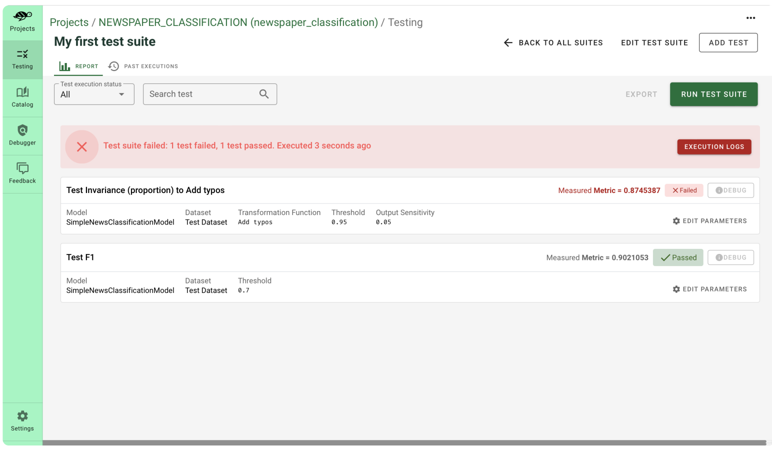
4. Debug your model and dataset

With the Debugger you can conveniently analyse each prediction. You can check, if it is correct or not, and analyse the features contribution to the prediction. You can tweak and change inputs to see how it affects output.

Debugger page. We can view the predicted probabilities of the classes, correctness of the prediction and the words that contributed the most to the output.

5. Testing

As was stated above, the Giskard UI provides a convenient way to execute test suites. We can modify the uploaded suite by adding new types of tests or create a new one. Giskard provides a wide variety of model's performance tests.



Testing page. Example of a test suite wherein one test has failed and another passed.

-

Fine-tuning adapts a pre-trained model to the new data without training it from scratch. After fine-tuning, you may want to check the model performance and behaviour. This can be done by uploading a fine-tuned model to the Giskard Server to perform inspection and testing routines, described in the previous section.

In this example, we are going to prepare and upload the XLM-ROBERTA model, which is fine-tuned on the SST-2 binary text classification dataset.

1. Data preparation

```
from enum import Enum

import torch
import numpy as np
import pandas as pd
import torch.nn as nn
from torch.optim import AdamW
import torchtext.transforms as T
import torchtext.functional as F
from torchtext.datasets import SST2
```

```
RobertaClassificationHead, XLMR_BASE_ENCODER


# URLs.
XLM_ROBERTA_VOCAB_URL = r"https://download.pytorch.org/models/text/xlmr.vocab.pt"
XLM_ROBERTA_MODEL_URL = r"https://download.pytorch.org/models/text/xlmr.sentencepiece.bpe.model"

# Constants.
BATCH_SIZE = 16
NUM_CLASSES = 2
MAX_SEQ_LEN = 256
MODEL_INPUT_DIM = 768

TEXT_FEATURE_NAME = "text"
TARGET_FEATURE_NAME = "label"

DEVICE = torch.device("cpu")


class Tokens(int, Enum):
    END = 2
    BEGIN = 0
    PADDING = 1


# Load data-pipes and transform them into data-loaders.
train_datapipe = SST2(split="train")
valid_datapipe = SST2(split="dev")

vocab = load_state_dict_from_url(XLM_ROBERTA_VOCAB_URL)
text_transform = T.Sequential(
    T.SentencePieceTokenizer(XLM_ROBERTA_MODEL_URL),
    T.VocabTransform(vocab),
```

```python


def apply_transform(x):
    return text_transform(x[0]), x[1]


def get_loader_from_pipe(datapipe, **kwargs):
    datapipe = datapipe.map(apply_transform)
    datapipe = datapipe.batch(BATCH_SIZE)
    datapipe = datapipe.rows2columnar(["token_ids", "target"])
    dataloader = DataLoader(datapipe, batch_size=None, shuffle=kwargs.get("shuffle", True))
    return dataloader


train_dataloader = get_loader_from_pipe(train_datapipe)
valid_dataloader = get_loader_from_pipe(valid_datapipe, shuffle=False)
```

giskard_sst2_01_data_preparation.py hosted with ❤️ by GitHub view raw

2. Wrap data with Giskard

```python
from giskard import Dataset


raw_data = pd.DataFrame(valid_datapipe, columns=[TEXT_FEATURE_NAME, TARGET_FEATURE_NAME])
wrapped_data = Dataset(raw_data,
                       name="sst2-dev-data",
                       target="label",
                       column_types={TEXT_FEATURE_NAME: "text", TARGET_FEATURE_NAME: "category"})
```

```
# Define model.
classifier_head = RobertaClassificationHead(num_classes=NUM_CLASSES, input_dim=MODEL_INPUT_DIM)
model = XLMR_BASE_ENCODER.get_model(head=classifier_head).to(DEVICE)
softmax = nn.Softmax(dim=1)

# Perform model fine-tuning.
criteria = nn.CrossEntropyLoss()
optim = AdamW(model.parameters(), lr=1e-5)


def train_iter(input_features, ground_truth):
    optim.zero_grad()
    prediction = model(input_features)
    train_loss = criteria(prediction, ground_truth)
    train_loss.backward()
    optim.step()

    print(train_loss.item())


NUM_EPOCHS = 5
NUM_ITERS = 100


def train():
    for epoch in range(NUM_EPOCHS):
        model.train()
        for idx, batch in enumerate(train_dataloader):
            print(f"Batch {idx}")

            # Perform training step.
            inputs = F.to_tensor(batch["token_ids"], padding_value=Tokens.PADDING).to(DEVICE)
            targets = torch.tensor(batch["target"]).to(DEVICE)
```

```
        print(f"Epoch = [{epoch}], loss = [{loss}], accuracy = [{accuracy}]")


def valid_iter(input_features, ground_truth):
    output = model(input_features)
    valid_loss = float(criteria(output, ground_truth).item())
    return valid_loss, (output.argmax(1) == ground_truth).type(torch.float).sum().item()


def evaluate():
    model.eval()

    total_loss = 0
    correct_predictions = 0
    total_predictions = 0
    counter = 0

    with torch.no_grad():
        for valid_batch in valid_dataloader:
            input_features = F.to_tensor(valid_batch["token_ids"], padding_value=Tokens.PADDING).to(DEVICE)
            ground_truth = torch.tensor(valid_batch["target"]).to(DEVICE)
            valid_loss, predictions = valid_iter(input_features, ground_truth)
            total_loss += valid_loss
            correct_predictions += predictions
            total_predictions += len(ground_truth)
            counter += 1

    return total_loss / counter, correct_predictions / total_predictions


train()
```

```python
def prediction_function(df):
    dataloader = get_loader_from_pipe(IterableWrapper(df[TEXT_FEATURE_NAME]), shuffle=False)
    predictions = list()

    model.eval()
    with torch.no_grad():
        for batch in dataloader:
            input_features = F.to_tensor(batch["token_ids"], padding_value=Tokens.PADDING).to(DEVICE)
            logits = model(input_features)
            probs = softmax(logits).detach().cpu().numpy()
            predictions.extend(probs)

    return np.array(predictions)
```

giskard_sst2_04_define_prediction_function.py view raw
hosted with ❤️ by GitHub

5. Wrap model with Giskard

```python
from giskard import Model


wrapped_model = Model(prediction_function,
                      model_type="classification",
                      name="xlm-roberta",
                      feature_names=[TEXT_FEATURE_NAME],
                      classification_labels=[0, 1])

# Validate wrapped model.
wrapped_model.predict(wrapped_data)
```

giskard_sst2_05_model_wrapping.py hosted with ❤️ by GitHub view raw performing the model scan and test-suite generation, we are ready to upload the fine-tuned model and the dataset to the Giskard Server.

```python
from giskard import GiskardClient


# Uploading the test suite will automatically save the model, dataset, tests, slicing & transformation functions inside the Giskard UI server
# Create a Giskard client after having install the Giskard server (see documentation)
token = ""  # Find it in Settings in the Giskard server

client = GiskardClient(
    url="http://localhost:19000",  # URL of your Giskard instance
    token=token
)

my_project = client.create_project("project_id", "PROJECT_NAME", "Project description.")

# Upload to the current project ✉
test_suite.upload(client, "project_id")
```

giskard_twitter_sentiment_07_artifacts_upload.py view raw
hosted with ❤️ by GitHub

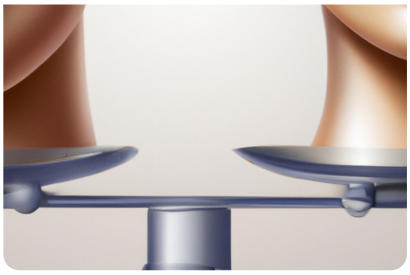
✅ Conclusion

demonstrated a full pipeline from creating and scanning model to debugging and testing it on the UI. We hope that you find this article helpful. Happy testing!

You will also like

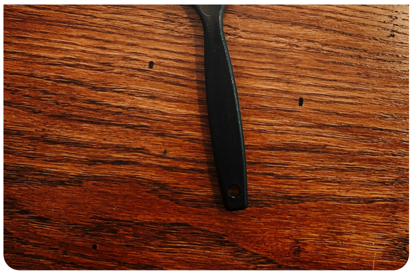
How to deploy a robust HuggingFace model for sentiment analysis into production?

Princy Pappachan



How to test the fairness of ML models? The 80% rule to measure the disparate impact

Rabah Abdul Khalek



How to test ML models? #1 👉 Introduction

Jean-Marie John-Mathews, Ph.D.

ensure the quality of all AI
models.

AI Scan
AI Test
Quickstart
Benefits

News
Tutorials
Community
Documentation

Team
Investors
Jobs
Contact us