ugtm Documentation

Release v2.0.0

Helena A. Gaspar

Feb 22, 2021

Get started here

1 Overview 1

2 Installation 3

3 eGTM: GTM transformer 5

4 eGTC: GTM classifier 7

5 eGTR: GTM regressor 11

6 API Reference 15

7 Visualization examples 19

8 Classification examples 27

9 Regression examples 31

10 tutorial 33

11 Glossary 39

12 Links & references 41

Index 43

CHAPTER 1

Overview

Generative topographic mapping (GTM) is a probabilistic dimensionality reduction algorithm introduced by Bishop, Svensen and Williams, which can also be used for classification and regression using class maps or activity landscapes:

[graph] ugtm v2.0 provides sklearn-compatible GTM transformer (eGTM), GTM classifier (eGTC) and GTM regressor (eGTR):

```
from ugtm import eGTM, eGTC, eGTR
import numpy as np

# Dummy train and test
X_train = np.random.randn(100, 50)
X_test = np.random.randn(50, 50)
y_train = np.random.choice([1, 2, 3], size=100)

# GTM transformer
transformed = eGTM().fit(X_train).transform(X_test)

# Predict new labels using GTM classifier (GTC)
predicted_labels = eGTC().fit(X_train, y_train).predict(X_test)

# Predict new continuous outcomes using GTM regressor (GTR)
predicted_labels = eGTR().fit(X_train, y_train).predict(X_test)
```

CHAPTER 2

Installation

2.1 Prerequisites

ugtm requires Python 2.7 or + (tested on Python 3.4.6 and Python 2.7.14), with following packages:

- scikit-learn>=0.20
- numpy>=1.14.5
- matplotlib>=2.2.2
- scipy>=0.19.1
- mpld3>=0.3
- jinja2>=2.10

2.2 pip installation

Install using pip in the command line:

```
pip install ugtm
```

If this does not work, try upgrading packages:

```
sudo pip install --upgrade pip numpy scikit-learn matplotlib scipy mpld3 jinja2
```

2.3 Using anaconda

Example of anaconda virtual env "p2" for python 2.7.14:

```
conda create -n p2 python=2.7.14 numpy=1.14.5 \
scikit-learn=0.20 matplotlib=2.2.2 \
scipy=0.19.1 mpld3=0.3 jinja2=2.10

# Activate virtual env
source activate p2

# Install package
pip install ugtm
```

Example of anaconda virtual env p3 for python 3.6.6:

```
conda create -n p3 python=3.6.6 numpy=1.14.5 \
scikit-learn=0.20 matplotlib=2.2.2 \
scipy=0.19.1 mpld3=0.3 jinja2=2.10

# Activate virtual env
source activate p3

# Install package
pip install ugtm
```

2.4 Import package

In python console, import ugtm package:

```
import ugtm
```

CHAPTER 3

eGTM: GTM transformer

3.1 Run GTM

`eGTM` is a sklearn-compatible GTM transformer. Similarly to PCA or t-SNE, eGTM reduces the dimensionality from n_dimensions to 2 dimensions. To generate mean GTM 2D projections:

```
from ugtm import eGTM
import numpy as np

X_train = np.random.randn(100, 50)
X_test = np.random.randn(50, 50)

# Fit GTM on X_train and get 2D projections for X_test
transformed = eGTM().fit(X_train).transform(X_test)
```

The default output of eGTM.transform is the mean GTM projection. For other data representations (modes, responsibilities), see `transform()`. Example of generating responsibilities (posterior probabilities for each node on the manifold) and reverse-mapping them to the input space:

```
from ugtm import eGTM
import numpy as np

X_train = np.random.randn(100, 50)

gtm_model = eGTM(model='responsibilities').fit(X_train)
transformed = gtm_model.transform(X_train)
inverse_transform = gtm_model.inverse_transform(transformed)
```

3.2 Visualize projection

Visualization demo using altair https://altair-viz.github.io:

```
from ugtm import eGTM
import numpy as np
import altair as alt
import pandas as pd

X_train = np.random.randn(100, 50)
X_test = np.random.randn(50, 50)

transformed = eGTM().fit(X_train).transform(X_test)

df = pd.DataFrame(transformed, columns=["x1", "x2"])
alt.Chart(df).mark_point().encode(
x='x1',y='x2',
tooltip=["x1", "x2"]
).properties(title="GTM projection of X_test").interactive()
```

[graph]

CHAPTER 4

eGTC: GTM classifier

4.1 Run eGTC

`eGTC` is a sklearn-compatible GTM classifier. Similarly to PCA or t-SNE, GTM reduces the dimensionality from n_dimensions to 2 dimensions. GTC uses a GTM class map to predict labels for new data (cf. `classMap()`). Two algorithms are available: the bayesian classifier GTC (`uGTC`) or the nearest node classifier (`uGTCnn`). The following example uses the iris dataset:

```
from ugtm import eGTC
from sklearn import datasets
from sklearn import preprocessing
from sklearn import decomposition
from sklearn import metrics
from sklearn import model_selection

iris = datasets.load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = model_selection.train_test_split(
X, y, test_size=0.33, random_state=42)

# optional preprocessing
scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# Predict labels for X_test
gtc = eGTC()
gtc = gtc.fit(X_train,y_train)
y_pred = gtc.predict(X_test)

# Print score
print(metrics.matthews_corrcoef(y_test,y_pred))
```

4.2 Visualize class map

The GTC algorithm is based on a classification map, discretized into a grid of nodes, which are colored by predicted label. To each node is associated class probabilities:

```
from ugtm import eGTM, eGTC
import numpy as np
import altair as alt
import pandas as pd
from sklearn import datasets
from sklearn import preprocessing
from sklearn import decomposition
from sklearn import metrics
from sklearn import model_selection

iris = datasets.load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = model_selection.train_test_split(
X, y, test_size=0.33, random_state=42)

# optional preprocessing
std = preprocessing.StandardScaler()
X_train = std.fit(X_train).transform(X_train)

# Construct class map
gtc = eGTC()
gtc = gtc.fit(X_train,y_train)

dfclassmap = pd.DataFrame(gtc.optimizedModel.matX, columns=["x1", "x2"])
dfclassmap["predicted_node_label"] = iris.target_names[gtc.node_label]
dfclassmap["probability_of_predominant_class"] = np.max(gtc.node_probabilities,axis=1)

# Classification map
alt.Chart(dfclassmap).mark_square().encode(
    x='x1',
    y='x2',
    color='predicted_node_label:N',
    size=alt.value(50),
    opacity='probability_of_predominant_class:Q',
    tooltip=['x1','x2', 'predicted_node_label:N', 'probability_of_predominant_class:Q
↪']
).properties(title = "Class map", width = 200, height = 200)
```

[graph]

4.3 Visualize predicted vs real labels

Visualize predicted vs real labels using the iris dataset and altair:

```
from ugtm import eGTM, eGTC
import numpy as np
import altair as alt
import pandas as pd
```

(continues on next page)

(continued from previous page)

```
from sklearn import datasets
from sklearn import preprocessing
from sklearn import decomposition
from sklearn import model_selection
from sklearn.metrics import confusion_matrix

iris = datasets.load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = model_selection.train_test_split(
X, y, test_size=0.33, random_state=42)

# optional preprocessing
scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# Predict labels for X_test
gtc = eGTC()
gtc = gtc.fit(X_train,y_train)
y_pred = gtc.predict(X_test)

# Get GTM transform for X_test
transformed = eGTM().fit(X_train).transform(X_test)

df = pd.DataFrame(transformed, columns=["x1", "x2"])
df["predicted_label"] = iris.target_names[y_pred]
df["true_label"] = iris.target_names[y_test]
df["probability_of_predominant_class"] = np.max(gtc.posteriors,axis=1)

# Projection of X_test colored by predicted label
chart1 = alt.Chart().mark_circle().encode(
    x='x1',y='x2',
    size=alt.value(100),
    color=alt.Color("predicted_label:N",
           legend=alt.Legend(title="label")),
    opacity="probability_of_predominant_class:Q",
    tooltip=["x1", "x2", "predicted_label:N",
             "true_label:N", "probability_of_predominant_class:Q"]
).properties(title="Pedicted labels", width=200, height=200).interactive()

# Projection of X_test colored by true_label
chart2 = alt.Chart().mark_circle().encode(
    x='x1', y='x2',
    color=alt.Color("true_label:N",
                    legend=alt.Legend(title="label")),
    size=alt.value(100),
    tooltip=["x1", "x2", "predicted_label:N",
             "true_label:N", "probability_of_predominant_class:Q"]
).properties(title="True_labels", width=200, height=200).interactive()


alt.hconcat(chart1, chart2, data=df)
```

[graph]

4.4 Parameter optimization

GridSearchCV can be used with eGTC for parameter optimization:

```
from ugtm import eGTC
import numpy as np
from sklearn.model_selection import GridSearchCV

# Dummy train and test
X_train = np.random.randn(100, 50)
X_test = np.random.randn(50, 50)
y_train = np.random.choice([1, 2, 3], size=100)

# Parameters to tune
tuned_params = {'regul': [0.0001, 0.001, 0.01],
                's': [0.1, 0.2, 0.3],
                'k': [16],
                'm': [4]}

# GTM classifier (GTC), bayesian
gs = GridSearchCV(eGTC(), tuned_params, cv=3, iid=False, scoring='accuracy')
gs.fit(X_train, y_train)
print(gs.best_params_)
```

CHAPTER 5

eGTR: GTM regressor

5.1 Run eGTR

`eGTR` is a sklearn-compatible GTM regressor. Similarly to PCA or t-SNE, GTM reduces the dimensionality from n_dimensions to 2 dimensions. GTR uses a GTM class map to predict labels for new data (cf. `landscape()`). The following example uses the iris dataset:

```
from ugtm import eGTR
from sklearn import datasets
from sklearn import preprocessing
from sklearn import decomposition
from sklearn import model_selection

boston = datasets.load_boston()
X = boston.data
y = boston.target

X_train, X_test, y_train, y_test = model_selection.train_test_split(
X, y, test_size=0.33, random_state=42)

# optional preprocessing
scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# Predict labels for X_test
gtr = eGTR()
gtr = gtr.fit(X_train,y_train)
y_pred = gtr.predict(X_test)
```

5.2 Visualize activity landscape

The GTR algorithm is based on an activity landscape. This landscape is discretized into a grid of nodes, which can be colored by predicted label. This visualization uses the python package altair:

```
from ugtm import eGTR, eGTM
import numpy as np
import altair as alt
import pandas as pd
from sklearn import datasets
from sklearn import preprocessing
from sklearn import decomposition
from sklearn import metrics
from sklearn import model_selection

boston = datasets.load_boston()
X = boston.data
y = boston.target

X_train, X_test, y_train, y_test = model_selection.train_test_split(
X, y, test_size=0.33, random_state=42)

# optional preprocessing
std = preprocessing.StandardScaler()
X_train = std.fit(X_train).transform(X_train)

# Construct activity landscape
gtr = eGTR()
gtr = gtr.fit(X_train,y_train)

dfclassmap = pd.DataFrame(gtr.optimizedModel.matX, columns=["x1", "x2"])
dfclassmap["predicted_node_label"] = gtr.node_label

# Classification map
alt.Chart(dfclassmap).mark_square().encode(
    x='x1',
    y='x2',
    color=alt.Color('predicted_node_label:Q',
                    scale=alt.Scale(scheme='greenblue'),
                    legend=alt.Legend(title="Boston house prices")),
    size=alt.value(50),
    tooltip=['x1','x2', 'predicted_node_label:Q']
).properties(title = "Activity landscape", width = 200, height = 200)
```

[graph]

5.3 Visualize predicted vs real labels

This visualization uses the python package altair:

```
from ugtm import eGTM, eGTR
import numpy as np
import altair as alt
import pandas as pd
from sklearn import datasets
```

(continues on next page)

(continued from previous page)

```
from sklearn import preprocessing
from sklearn import decomposition
from sklearn import metrics
from sklearn import model_selection

boston = datasets.load_boston()
X = boston.data
y = boston.target

X_train, X_test, y_train, y_test = model_selection.train_test_split(
X, y, test_size=0.33, random_state=42)

# optional preprocessing
scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# Predict labels for X_test
gtr = eGTR()
gtr = gtr.fit(X_train,y_train)
y_pred = gtr.predict(X_test)

# Get GTM transform for X_test
transformed = eGTM().fit(X_train).transform(X_test)

df = pd.DataFrame(transformed, columns=["x1", "x2"])
df["predicted_label"] = y_pred
df["true_label"] = y_test

chart1 = alt.Chart(df).mark_point().encode(
x='x1',y='x2',
color=alt.Color("predicted_label:Q",scale=alt.Scale(scheme='greenblue'),
                 legend=alt.Legend(title="Boston house prices")),
tooltip=["x1", "x2", "predicted_label:Q", "true_label:Q"]
).properties(title="Pedicted labels", width=200, height=200).interactive()

chart2 = alt.Chart(df).mark_point().encode(
x='x1',y='x2',
color=alt.Color("true_label:Q",scale=alt.Scale(scheme='greenblue'),
                legend=alt.Legend(title="Boston house prices")),
tooltip=["x1", "x2", "predicted_label:Q", "true_label:Q"]
).properties(title="True labels", width=200, height=200).interactive()

alt.hconcat(chart1, chart2)
```

[graph]

5.4 Parameter optimization

GridSearchCV from sklearn can be used with eGTC for parameter optimization:

```
from ugtm import eGTR
import numpy as np
from sklearn.model_selection import GridSearchCV
```

(continues on next page)

(continued from previous page)

```
# Dummy train and test
X_train = np.random.randn(100, 50)
X_test = np.random.randn(50, 50)
y_train = np.random.choice([1, 2, 3], size=100)

# Parameters to tune
tuned_params = {'regul': [0.0001, 0.001, 0.01],
                's': [0.1, 0.2, 0.3],
                'k': [16],
                'm': [4]}

# GTM classifier (GTR)
gs = GridSearchCV(eGTR(), tuned_params, cv=3, iid=False, scoring='r2')
gs.fit(X_train, y_train)
print(gs.best_params_)
```

CHAPTER 6

API Reference

API reference of ugtm. This documentation was generated automatically from docstrings.

6.1 Modules

ugtm	ugtm: a python package for Generative Topographic Mapping (GTM)

6.1.1 ugtm

ugtm: a python package for Generative Topographic Mapping (GTM)

Submodules

ugtm_sklearn	GTM transformer, classifier and regressor compatible with sklearn
ugtm_gtm	Functions to run GTM models.
ugtm_kgtm	Functions to initialize and optimize kernel GTM models.
ugtm_classes	Defines classes for initial and optimized GTM model.
ugtm_plot	ugtm plot functions.
ugtm_landscape	Builds continuous GTM class maps or landscapes using labels or activities.
ugtm_predictions	GTC (GTM classification) and GTR (GTM regression)
ugtm_crossvalidate	Cross-validation support for GTC and GTR models (also SVM and PCA).
ugtm_preprocess	Preprocessing operations (mostly using scikit-learn functions).

ugtm.ugtm_sklearn

GTM transformer, classifier and regressor compatible with sklearn

Classes

eGTC([k, m, s, regul, random_state, niter, ...])	eGTC : GTC Bayesian classifier for sklearn pipelines.
eGTCnn([k, m, s, regul, random_state, ...])	eGTCnn: GTC nearest node classifier for sklearn pipelines.
eGTM([k, m, s, regul, random_state, niter, ...])	eGTM: GTM Transformer for sklearn pipeline.
eGTR([k, m, s, regul, random_state, niter, ...])	eGTR: GTM nearest node(s) regressor for sklearn pipelines.

ugtm.ugtm_gtm

Functions to run GTM models.

Functions

initialize(data, k, m, s[, random_state])	Initializes a GTM model.
optimize(data, initialModel, regul, niter[, ...])	Optimizes a GTM model.
projection(optimizedModel, new_data)	Project test set on optimized GTM model.
runGTM(data[, k, m, s, regul, doPCA, ...])	Run GTM (wrapper for initialize + optimize).
transform(optimizedModel, train, test[, ...])	Preprocess and project test set on optimized GTM model.

ugtm.ugtm_kgtm

Functions to initialize and optimize kernel GTM models.

Functions

initializeKernel(data, k, m, s, maxdim[, ...])	Initializes a kernel GTM (kGTM) model.
optimizeKernel(data, initialModel, regul, niter)	Optimizes a kGTM model.
runkGTM(data[, k, m, s, regul, maxdim, ...])	Run kGTM algorithm (wrapper for initialize + optimize).

ugtm.ugtm_classes

Defines classes for initial and optimized GTM model.

Classes

InitialGTM(matX, matM, n_nodes, ...)	Class for initial GTM model.
OptimizedGTM(matW, matY, matP, matR, ...)	Class for optimized GTM model.

Continued on next page

Table 6 – continued from previous page

ReturnU(matU, betaInv)	

ugtm.ugtm_plot

ugtm plot functions.

Functions

plot(coordinates[, labels, title, output, ...])	Simple plotting function.
plotClassMap(optimizedModel, labels[, ...])	Plots GTM class map.
plotClassMapNoPoints(optimizedModel, labels)	Plots GTM class map without 2D representations.
plotLandscape(optimizedModel, labels[, ...])	Plots GTM landscape.
plotLandscapeNoPoints(optimizedModel, labels)	Plots GTM landscape without 2D representations.
plotMultiPanelGTM(optimizedModel, labels[, ...])	Multipanel visualization for GTM object - PDF output.
plot_html(coordinates[, labels, ids, title, ...])	Simple plotting function: HTML output.
plot_html_GTM(optimizedModel[, labels, ids, ...])	Plotting function for GTM object - HTML output.
plot_html_GTM_projection(optimizedModel, ...)	Returns a GTM landscape with projected data points - HTML output.
plot_pdf(*args, **kwargs)	

Classes

NumpyEncoder([skipkeys, ensure_ascii, ...])	

ugtm.ugtm_landscape

Builds continuous GTM class maps or landscapes using labels or activities.

Functions

classMap(optimizedModel, activity[, prior])	Computes GTM class map based on discrete activities (= discrete labels)
landscape(optimizedModel, activity)	Computes GTM landscapes based on activities (= continuous labels).

Classes

ClassMap(nodeClassP, nodeClassT, ...)	Class for ClassMap: Bayesian classification model for each GTM node.

ugtm.ugtm_predictions

GTC (GTM classification) and GTR (GTM regression)

Functions

GTC(train, labels, test[, k, m, s, regul, ...])	Run GTC (GTM classification): Bayes or nearest node algorithm.
GTR(train, labels, test[, k, m, s, regul, ...])	Run GTR (GTM nearest node(s) regression).
advancedGTC(train, labels, test[, ...])	Run GTC (GTM classification): advanced Bayes
advancedPredictBayes(optimizedModel, labels, ...)	Bayesian GTM classifier: complete model
predictBayes(optimizedModel, labels, new_data)	Bayesian GTM classifier (GTC Bayes).
predictNN(optimizedModel, labels, new_data)	GTM nearest node(s) classification or regression.
predictNNSimple(train, test, labels[, ...])	Nearest neighbor(s) classification or regression.
printClassPredictions(prediction, output)	Print output of advancedPredictBayes().

ugtm.ugtm_crossvalidate

Cross-validation support for GTC and GTR models (also SVM and PCA).

Functions

crossvalidateGTC(data, labels[, k, m, s, ...])	Cross-validate GTC model.
crossvalidateGTR(data, labels[, k, m, s, ...])	Cross-validate GTR model.
crossvalidatePCAC(data, labels[, ...])	Cross-validate PCA kNN classification model.
crossvalidatePCAR(data, labels[, ...])	Cross-validate PCA kNN regression model.
crossvalidateSVC(data, labels[, C, doPCA, ...])	Cross-validate SVC model.
crossvalidateSVCrbf(data, labels[, C, ...])	Cross-validate SVC model with RBF kernel.
crossvalidateSVR(data, labels[, C, epsilon, ...])	Cross-validate SVR model with linear kernel.
whichExperiment(data, labels, args[, discrete])	

ugtm.ugtm_preprocess

Preprocessing operations (mostly using scikit-learn functions).

Functions

chooseKernel(data[, kerneltype])	Kernalize data (uses sklearn)
pcaPreprocess(data[, doPCA, n_components, ...])	Preprocess data using PCA.
processTrainTest(train, test, doPCA, ...[, ...])	Preprocess train and test data using PCA.

Classes

ProcessedTrainTest(train, test)	Class for processed train and test set.

CHAPTER 7

Visualization examples

GTM visualization examples from following datasets:

- S-curve (from sklearn s-curve example)
- Severed sphere (from sklearn severed sphere example)
- Hand-written digits (from sklearn hand-written digits example)

7.1 S-curve

```
from ugtm import eGTM,eGTR
import numpy as np
import altair as alt
import pandas as pd
from sklearn import datasets
from sklearn import metrics
from sklearn import model_selection
from sklearn import manifold

X,y = datasets.make_s_curve(n_samples=1000, random_state=0)
man = manifold.TSNE(n_components=2, init='pca', random_state=0)
tsne = man.fit_transform(X)
man = manifold.MDS(max_iter=100, n_init=1, random_state=0)
mds = man.fit_transform(X)
man = manifold.LocallyLinearEmbedding(n_neighbors=20, n_components=2,
                                      eigen_solver='auto',
                                      method="standard",
                                      random_state=0)
lle = man.fit_transform(X)

# Construct GTM
gtm_means = eGTM(m=2, model="means").fit_transform(X)
gtm_modes = eGTM(m=2, model="modes").fit_transform(X)
```

(continues on next page)

(continued from previous page)

```
dgtm_modes = pd.DataFrame(gtm_modes, columns=["x1", "x2"])
dgtm_modes["label"] = y

gtm_modes = alt.Chart(dgtm_modes).mark_circle().encode(
    x='x1',
    y='x2',
    color=alt.Color('label:Q',
                    scale=alt.Scale(scheme='viridis')),
    size=alt.value(50),
    tooltip=['x1','x2','label:Q']
).properties(title = "GTM (modes)", width = 100, height = 100)

dgtm_means = pd.DataFrame(gtm_means, columns=["x1", "x2"])
dgtm_means["label"] = y

gtm_means = alt.Chart(dgtm_means).mark_circle().encode(
    x='x1',
    y='x2',
    color=alt.Color('label:Q',
                    scale=alt.Scale(scheme='viridis')),
    size=alt.value(50),
    tooltip=['x1','x2','label:Q']
).properties(title = "GTM (means)", width = 100, height = 100)

#Construct activity landscape
gtr = eGTR(m=2)
gtr = gtr.fit(X,y)

dfclassmap = pd.DataFrame(gtr.optimizedModel.matX, columns=["x1", "x2"])
dfclassmap["label"] = gtr.node_label

# Classification map
gtr = alt.Chart(dfclassmap).mark_square().encode(
    x='x1',
    y='x2',
    color=alt.Color('label:Q',
                    scale=alt.Scale(scheme='viridis')),
    size=alt.value(50),
    tooltip=['x1','x2', 'label:Q'],
    #opacity='density'
).properties(title = "GTM landscape",width = 100, height = 100)

dtsne = pd.DataFrame(tsne, columns=["x1", "x2"])
dmds = pd.DataFrame(mds, columns=["x1", "x2"])
dlle = pd.DataFrame(lle, columns=["x1", "x2"])
dtsne["label"] = y
dmds["label"] = y
dlle["label"] = y

tsne = alt.Chart(dtsne).mark_circle().encode(
    x='x1',
    y='x2',
    color=alt.Color('label:Q',
                    scale=alt.Scale(scheme='viridis')),
    size=alt.value(50),
    tooltip=['x1','x2','label:Q']
```

(continues on next page)

(continued from previous page)

```
).properties(title = "t-SNE", width = 100, height = 100)

mds = alt.Chart(dmds).mark_circle().encode(
    x='x1',
    y='x2',
    color=alt.Color('label:Q',
                    scale=alt.Scale(scheme='viridis')),
    size=alt.value(50),
    tooltip=['x1','x2','label:Q']
).properties(title = "MDS", width = 100, height = 100)

lle = alt.Chart(dlle).mark_circle().encode(
    x='x1',
    y='x2',
    color=alt.Color('label:Q',
                    scale=alt.Scale(scheme='viridis')),
    size=alt.value(50),
    tooltip=['x1','x2','label:Q']
).properties(title = "LLE", width = 100, height = 100)


gtm = gtm_means | gtm_modes | gtr
others = tsne | mds | lle

alt.vconcat(gtm, others)
```

[graph]

7.2 Severed sphere

```
from ugtm import eGTM,eGTR
import numpy as np
import altair as alt
import pandas as pd
from sklearn import datasets
from sklearn import metrics
from sklearn import model_selection
from sklearn import manifold
from sklearn.utils import check_random_state

random_state = check_random_state(0)
p = random_state.rand(1000) * (2 * np.pi - 0.55)
t = random_state.rand(1000) * np.pi

# Sever the poles from the sphere.
indices = ((t < (np.pi - (np.pi / 8))) & (t > ((np.pi / 8))))
x, y, z = np.sin(t[indices]) * np.cos(p[indices]), \
    np.sin(t[indices]) * np.sin(p[indices]), \
    np.cos(t[indices])

X = np.array([x, y, z]).T

y = p[indices]
```

(continues on next page)

(continued from previous page)

```
man = manifold.TSNE(n_components=2, init='pca', random_state=0)
tsne = man.fit_transform(X)
man = manifold.MDS(max_iter=100, n_init=1, random_state=0)
mds = man.fit_transform(X)
man = manifold.LocallyLinearEmbedding(n_neighbors=10, n_components=2,
                                      eigen_solver='auto',
                                      method="standard",
                                      random_state=0)
lle = man.fit_transform(X)

# Construct GTM
gtm_means = eGTM(m=2, model="means").fit_transform(X)
gtm_modes = eGTM(m=2, model="modes").fit_transform(X)

dgtm_modes = pd.DataFrame(gtm_modes, columns=["x1", "x2"])
dgtm_modes["label"] = y

gtm_modes = alt.Chart(dgtm_modes).mark_circle().encode(
    x='x1',
    y='x2',
    color=alt.Color('label:Q',
                    scale=alt.Scale(scheme='viridis')),
    size=alt.value(50),
    tooltip=['x1','x2','label:Q']
).properties(title = "GTM (modes)", width = 100, height = 100)

dgtm_means = pd.DataFrame(gtm_means, columns=["x1", "x2"])
dgtm_means["label"] = y

gtm_means = alt.Chart(dgtm_means).mark_circle().encode(
    x='x1',
    y='x2',
    color=alt.Color('label:Q',
                    scale=alt.Scale(scheme='viridis')),
    size=alt.value(50),
    tooltip=['x1','x2','label:Q']
).properties(title = "GTM (means)", width = 100, height = 100)

#Construct activity landscape
gtr = eGTR(m=2)
gtr = gtr.fit(X,y)

dfclassmap = pd.DataFrame(gtr.optimizedModel.matX, columns=["x1", "x2"])
dfclassmap["label"] = gtr.node_label

# Classification map
gtr = alt.Chart(dfclassmap).mark_square().encode(
    x='x1',
    y='x2',
    color=alt.Color('label:Q',
                    scale=alt.Scale(scheme='viridis')),
    size=alt.value(50),
    tooltip=['x1','x2', 'label:Q'],
    #opacity='density'
).properties(title = "GTM landscape",width = 100, height = 100)

dtsne = pd.DataFrame(tsne, columns=["x1", "x2"])
```

(continues on next page)

(continued from previous page)

```
dmds = pd.DataFrame(mds, columns=["x1", "x2"])
dlle = pd.DataFrame(lle, columns=["x1", "x2"])
dtsne["label"] = y
dmds["label"] = y
dlle["label"] = y

tsne = alt.Chart(dtsne).mark_circle().encode(
    x='x1',
    y='x2',
    color=alt.Color('label:Q',
                    scale=alt.Scale(scheme='viridis')),
    size=alt.value(50),
    tooltip=['x1','x2','label:Q']
).properties(title = "t-SNE", width = 100, height = 100)

mds = alt.Chart(dmds).mark_circle().encode(
    x='x1',
    y='x2',
    color=alt.Color('label:Q',
                    scale=alt.Scale(scheme='viridis')),
    size=alt.value(50),
    tooltip=['x1','x2','label:Q']
).properties(title = "MDS", width = 100, height = 100)

lle = alt.Chart(dlle).mark_circle().encode(
    x='x1',
    y='x2',
    color=alt.Color('label:Q',
                    scale=alt.Scale(scheme='viridis')),
    size=alt.value(50),
    tooltip=['x1','x2','label:Q']
).properties(title = "LLE", width = 100, height = 100)


gtm = gtm_means | gtm_modes | gtr
others = tsne | mds | lle

alt.vconcat(gtm, others)
```

[graph]

7.3 Hand-written digits

```
from ugtm import eGTM,eGTC
import numpy as np
import altair as alt
import pandas as pd
from sklearn import datasets
from sklearn import metrics
from sklearn import model_selection
from sklearn import manifold
from sklearn.utils import check_random_state
```

(continues on next page)

(continued from previous page)

```
digits = datasets.load_digits(n_class=6)
X = digits.data
y = digits.target

man = manifold.TSNE(n_components=2, init='pca', random_state=0)
tsne = man.fit_transform(X)
man = manifold.MDS(max_iter=100, n_init=1, random_state=0)
mds = man.fit_transform(X)
man = manifold.LocallyLinearEmbedding(n_neighbors=20, n_components=2,
                                      eigen_solver='auto',
                                      method="standard",
                                      random_state=0)
lle = man.fit_transform(X)

# Construct GTM
gtm_means = eGTM(m=2, model="means").fit_transform(X)
gtm_modes = eGTM(m=2, model="modes").fit_transform(X)

dgtm_modes = pd.DataFrame(gtm_modes, columns=["x1", "x2"])
dgtm_modes["label"] = y

gtm_modes = alt.Chart(dgtm_modes).mark_circle().encode(
    x='x1',
    y='x2',
    color=alt.Color('label:N',
                    scale=alt.Scale(scheme='viridis')),
    size=alt.value(50),
    tooltip=['x1','x2','label:N']
).properties(title = "GTM (modes)", width = 100, height = 100)

dgtm_means = pd.DataFrame(gtm_means, columns=["x1", "x2"])
dgtm_means["label"] = y

gtm_means = alt.Chart(dgtm_means).mark_circle().encode(
    x='x1',
    y='x2',
    color=alt.Color('label:N',
                    scale=alt.Scale(scheme='viridis')),
    size=alt.value(50),
    tooltip=['x1','x2','label:N']
).properties(title = "GTM (means)", width = 100, height = 100)

#Construct activity landscape
gtc = eGTC()
gtc = gtc.fit(X,y)

dfclassmap = pd.DataFrame(gtc.optimizedModel.matX, columns=["x1", "x2"])
dfclassmap["label"] = gtc.node_label

# Classification map
gtc = alt.Chart(dfclassmap).mark_square().encode(
    x='x1',
    y='x2',
    color=alt.Color('label:N',
                    scale=alt.Scale(scheme='viridis')),
    size=alt.value(50),
    tooltip=['x1','x2', 'label:N'],
```

(continues on next page)

(continued from previous page)

```
    #opacity='density'
).properties(title = "GTM class map",width = 100, height = 100)

dtsne = pd.DataFrame(tsne, columns=["x1", "x2"])
dmds = pd.DataFrame(mds, columns=["x1", "x2"])
dlle = pd.DataFrame(lle, columns=["x1", "x2"])
dtsne["label"] = digits.target_names[y]
dmds["label"] = digits.target_names[y]
dlle["label"] = digits.target_names[y]

tsne = alt.Chart(dtsne).mark_circle().encode(
    x='x1',
    y='x2',
    color=alt.Color('label:N',
                    scale=alt.Scale(scheme='viridis')),
    size=alt.value(50),
    tooltip=['x1','x2','label:N']
).properties(title = "t-SNE", width = 100, height = 100)

mds = alt.Chart(dmds).mark_circle().encode(
    x='x1',
    y='x2',
    color=alt.Color('label:N',
                    scale=alt.Scale(scheme='viridis')),
    size=alt.value(50),
    tooltip=['x1','x2','label:N']
).properties(title = "MDS", width = 100, height = 100)

lle = alt.Chart(dlle).mark_circle().encode(
    x='x1',
    y='x2',
    color=alt.Color('label:N',
                    scale=alt.Scale(scheme='viridis')),
    size=alt.value(50),
    tooltip=['x1','x2','label:N']
).properties(title = "LLE", width = 100, height = 100)


gtm = gtm_means | gtm_modes | gtc
others = tsne | mds | lle

alt.vconcat(gtm, others)
```

[graph]

CHAPTER 8

Classification examples

8.1 Breast cancer

We use the breast cancer wisconsin dataset loaded from sklearn, downloaded from https://goo.gl/U2Uwz2.

The variables are the following:

a. radius (mean of distances from center to points on the perimeter)
b. texture (standard deviation of gray-scale values)
c. perimeter
d. area
e. smoothness (local variation in radius lengths)
f. compactness (perimeter^2 / area - 1.0)
g. concavity (severity of concave portions of the contour)
h. concave points (number of concave portions of the contour)
i. symmetry
j. fractal dimension ("coastline approximation" - 1)

The target variable is the diagnosis (malignant/benign).

Example of parameter selection and cross-validation using GTM classification (GTC) and SVM classification (SVC):

```
from ugtm import eGTC
from sklearn.datasets import load_breast_cancer
import numpy as np
from sklearn import model_selection
from sklearn.metrics import balanced_accuracy_score
from sklearn.svm import SVC
from sklearn.metrics import classification_report
```

(continues on next page)

(continued from previous page)

```
data = load_breast_cancer()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = model_selection.train_test_split(
X, y, test_size=0.33, random_state=42, shuffle=True)

performances = {}


# GTM classifier (GTC), bayesian

tuned_params = {'regul': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100],
                's': [0.1, 0.2, 0.3],
                'k': [16],
                'm': [4]}

gs = model_selection.GridSearchCV(eGTC(), tuned_params, cv=3, iid=False, scoring=
↪'balanced_accuracy')

gs.fit(X_train, y_train)

# Returns best score and best parameters
print(gs.best_score_)
print(gs.best_params_)

# Test data using model built with best parameters
y_true, y_pred = y_test, gs.predict(X_test)
print(classification_report(y_true, y_pred))

# Record performance on test set
performances['gtc'] = balanced_accuracy_score(y_true, y_pred)


# SVM classifier (SVC)

tuned_params = {'C':[1,10,100,1000],
                'gamma':[1,0.1,0.001,0.0001],
                'kernel':['rbf']}

gs = model_selection.GridSearchCV(SVC(random_state=42), tuned_params, cv=3, iid=False,
↪ scoring='balanced_accuracy')

gs.fit(X_train, y_train)

# Returns best score and best parameters
print(gs.best_score_)
print(gs.best_params_)

# Test data using model built with best parameters
y_true, y_pred = y_test, gs.predict(X_test)
print(classification_report(y_true, y_pred))

# Record performance on test set
performances['svm'] = balanced_accuracy_score(y_test, y_pred)
```

(continues on next page)

(continued from previous page)

```
# Algorithm with best performance
max(performances.items(), key = lambda x: x[1])
```

CHAPTER 9

Regression examples

9.1 Wine quality

We use the wine quality dataset from http://archive.ics.uci.edu/.

Example of parameter selection and cross-validation using GTM regression (GTR) and SVM classification (SVR):

```
from ugtm import eGTR
import numpy as np
from numpy import sqrt
from sklearn import model_selection
from sklearn.metrics import mean_squared_error
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.dummy import DummyRegressor
import pandas as pd

# Load red wine data
url = "http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/
↪winequality-red.csv"
data = pd.read_csv(url,sep=";")
y = data['quality']
X = data.drop(labels='quality',axis=1)


X_train, X_test, y_train, y_test = model_selection.train_test_split(
X, y, test_size=0.10, random_state=42, shuffle=True, random_state=42)


std = StandardScaler().fit(X_train)
X_train = std.transform(X_train)
X_test = std.transform(X_test)

performances = {}
```

(continues on next page)

(continued from previous page)

```
# GTM classifier (GTR), bayesian

tuned_params = {'regul': [0.0001, 0.001, 0.01, 0.1, 1],
        's': [0.1, 0.2, 0.3],
        'k': [25],
        'm': [5]}

gs = model_selection.GridSearchCV(eGTR(), tuned_params, cv=3, iid=False, scoring='neg_
↪mean_squared_error')

gs.fit(X_train, y_train)

# Returns best score and best parameters
print(gs.best_score_)
print(gs.best_params_)

# Test data using model built with best parameters
y_true, y_pred = y_test, gs.predict(X_test)

# Record performance on test set (RMSE)
performances['gtr'] = sqrt(mean_squared_error(y_true, y_pred))

# SVM regressor (SVR)

tuned_params = {'C':[1,10,100,1000],
        'gamma':[1,0.1,0.001,0.0001],
        'kernel':['rbf']}

gs = model_selection.GridSearchCV(SVR(), tuned_params, cv=3, iid=False, scoring='neg_
↪mean_squared_error')

gs.fit(X_train, y_train)

# Returns best score and best parameters
print(gs.best_score_)
print(gs.best_params_)

# Test data using model built with best parameters
y_true, y_pred = y_test, gs.predict(X_test)

# Record performance on test set
performances['svm'] = sqrt(mean_squared_error(y_test, y_pred))

# Create a dummy regressor
dummy = DummyRegressor(strategy='mean')

# Train dummy regressor
dummy.fit(X_train, y_train)
y_true, y_pred = y_test, dummy.predict(X_test)

# Dummy performance
performances['dummy'] = sqrt(mean_squared_error(y_test, y_pred))
```

CHAPTER 10

tutorial

ugtm provides an implementation of GTM (Generative Topographic Mapping), kGTM (kernel Generative Topographic Mapping), GTM classification models (kNN, Bayes) and GTM regression models. ugtm also implements cross-validation options which can be used to compare GTM classification models to SVM classification models, and GTM regression models to SVM regression models. Typical usage:

```
#!/usr/bin/env python

import ugtm
import numpy as np

#generate sample data and labels: replace this with your own data
data=np.random.randn(100,50)
labels=np.random.choice([1,2],size=100)

#build GTM map
gtm=ugtm.runGTM(data=data,verbose=True)

#plot GTM map (html)
gtm.plot_html(output="out")
```

For installation instructions, cf. https://github.com/hagax8/ugtm

10.1 1. Import package

Import ugtm package, allowing to construct GTM and kernel GTM (kGTM) maps, GTM classification models, GTM regression models:

```
import ugtm
```

10.2 2. Construct and plot GTM maps (or kGTM maps)

A gtm object can be created by running the function runGTM on a dataset. Parameters for runGTM are: k = sqrt(number of nodes), m = sqrt(number of rbf centres), s = RBF width factor, regul = regularization coefficient. The number of iteration for the expectation-maximization algorithm is set to 200 by default. This is an example with random data:

```
import ugtm

#import numpy to generate random data
import numpy as np

#generate random data (independent variables x),
#discrete labels (dependent variable y),
#and continuous labels (dependent variable y),
#to experiment with categorical or continuous outcomes

train = np.random.randn(20,10)
test = np.random.randn(20,10)
labels=np.random.choice(["class1","class2"],size=20)
activity=np.random.randn(20,1)

#create a gtm object and write model
gtm = ugtm.runGTM(train)
gtm.write("testout1")

#run verbose
gtm = ugtm.runGTM(train, verbose=True)

#to run a kernel GTM model instead, run following:
gtm = ugtm.runkGTM(train, doKernel=True, kernel="linear")

#access coordinates (means or modes), and responsibilities of gtm object
gtm_coordinates = gtm.matMeans
gtm_modes = gtm.matModes
gtm_responsibilities = gtm.matR
```

10.3 3. Plot html maps

Call the plot_html() function on the gtm object:

```
#run model on train
gtm = ugtm.runGTM(train)

# ex. plot gtm object with landscape, html: labels are continuous
gtm.plot_html(output="testout10",labels=activity,discrete=False,pointsize=20)

# ex. plot gtm object with landscape, html: labels are discrete
gtm.plot_html(output="testout11",labels=labels,discrete=True,pointsize=20)

# ex. plot gtm object with landscape, html: labels are continuous
# no interpolation between nodes
gtm.plot_html(output="testout12",labels=activity,discrete=False,pointsize=20, \
              do_interpolate=False,ids=labels)
```

(continues on next page)

(continued from previous page)

```
# ex. plot gtm object with landscape, html: labels are discrete,
# no interpolation between nodes
gtm.plot_html(output="testout13",labels=labels,discrete=True,pointsize=20, \
              do_interpolate=False)
```

10.4 4. Plot pdf maps

Call the plot() function on the gtm object:

```
#run model on train
gtm = ugtm.runGTM(train)

# ex. plot gtm object, pdf: no labels
gtm.plot(output="testout6",pointsize=20)

# ex. plot gtm object with landscape, pdf: labels are discrete
gtm.plot(output="testout7",labels=labels,discrete=True,pointsize=20)

# ex. plot gtm object with landscape, pdf: labels are continuous
gtm.plot(output="testout8",labels=activity,discrete=False,pointsize=20)
```

10.5 5. Plot multipanel views (only if labels or activities are provided)

Call the plot_multipanel() function on the gtm object. This plots a general model view, showing means, modes, landscape with or without points. The plot_multipanel function only works if you have defined labels:

```
#run model on train
gtm = ugtm.runGTM(train)

# ex. with discrete labels and inter-node interpolation
gtm.plot_multipanel(output="testout2",labels=labels,discrete=True,pointsize=20)

# ex. with continuous labels and inter-node interpolation
gtm.plot_multipanel(output="testout3",labels=activity,discrete=False,pointsize=20)

# ex. with discrete labels and no inter-node interpolation
gtm.plot_multipanel(output="testout4",labels=labels,discrete=True,pointsize=20, \
                    do_interpolate=False)

# ex. with continuous labels and no inter-node interpolation
gtm.plot_multipanel(output="testout5",labels=activity,discrete=False,pointsize=20, \
                    do_interpolate=False)
```

10.6 6. Project new data onto existing GTM map

New data can be projected on the GTM map by using the transform() function, which takes as input the gtm model, a training and test set. The train set is then only used to perform data preprocessing on the test set based on the train (for example: apply the same PCA transformation to the train and test sets before running the algorithm):

```
#run model on train
gtm = ugtm.runGTM(train,doPCA=True)

#test new data (test)
transformed=ugtm.transform(optimizedModel=gtm,train=train,test=test,doPCA=True)

#plot transformed test (html)
transformed.plot_html(output="testout14",pointsize=20)

#plot transformed test (pdf)
transformed.plot(output="testout15",pointsize=20)

#plot transformed data on existing classification model,
#using training set labels
gtm.plot_html_projection(output="testout16",projections=transformed,\
                         labels=labels, \
                         discrete=True,pointsize=20)
```

10.7 7. Output predictions for a test set: GTM regression (GTR) and classification (GTC)

The GTR() function implements the GTM regression model (cf. references) and GTC() function a GTM classification model (cf. references):

```
#continuous labels (prediction by GTM regression model)
predicted=ugtm.GTR(train=train,test=test,labels=activity)

#discrete labels (prediction by GTM classification model)
predicted=ugtm.GTC(train=train,test=test,labels=labels)
```

10.8 8. Advanced GTM predictions with per-class probabilities

Per-class probabilities for a test set can be given by the advancedGTC() function (you can set the m, k, regul, s parameters just as with runGTM):

```
#get whole output model and label predictions for test set
predicted_model=ugtm.advancedGTC(train=train,test=test,labels=labels)

#write whole predicted model with per-class probabilities
ugtm.printClassPredictions(predicted_model,"testout17")
```

10.9 9. Crossvalidation experiments

Different crossvalidation experiments were implemented to compare GTC and GTR models to classical machine learning methods:

```
#crossvalidation experiment: GTM classification model implemented in ugtm,
#here: set hyperparameters s=1 and regul=1 (set to -1 to optimize)
ugtm.crossvalidateGTC(data=train,labels=labels,s=1,regul=1,n_repetitions=10,n_folds=5)
```

(continues on next page)

(continued from previous page)

```
#crossvalidation experiment: GTM regression model
ugtm.crossvalidateGTR(data=train,labels=activity,s=1,regul=1)

#you can also run the following functions to compare
#with other classification/regression algorithms:

#crossvalidation experiment, k-nearest neighbours classification
#on 2D PCA map with 7 neighbors (set to -1 to optimize number of neighbours)
ugtm.crossvalidatePCAC(data=train,labels=labels,n_neighbors=7)

#crossvalidation experiment, SVC rbf classification model (sklearn implementation):
ugtm.crossvalidateSVCrbf(data=train,labels=labels,C=1,gamma=1)

#crossvalidation experiment, linear SVC classification model (sklearn implementation):
ugtm.crossvalidateSVC(data=train,labels=labels,C=1)

#crossvalidation experiment, linear SVC regression model (sklearn implementation):
ugtm.crossvalidateSVR(data=train,labels=activity,C=1,epsilon=1)

#crossvalidation experiment, k-nearest neighbours regression on 2D PCA map with 7␣
↪neighbors:
ugtm.crossvalidatePCAR(data=train,labels=activity,n_neighbors=7)
```

CHAPTER 11

Glossary

- genindex

CHAPTER 12

Links & references

1. GTM algorithm by Bishop et al: https://www.microsoft.com/en-us/research/wp-content/uploads/1998/01/bishop-gtm-ncomp-98.pdf
2. kernel GTM: https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2010-44.pdf
3. GTM classification models: https://www.ncbi.nlm.nih.gov/pubmed/24320683
4. GTM regression models: https://www.ncbi.nlm.nih.gov/pubmed/27490381
5. Visualizations were realized using altair
6. Source code: https://github.com/hagax8/ugtm

Index

U

ugtm (*module*), 15
ugtm.ugtm_classes (*module*), 16
ugtm.ugtm_crossvalidate (*module*), 18
ugtm.ugtm_gtm (*module*), 16
ugtm.ugtm_kgtm (*module*), 16
ugtm.ugtm_landscape (*module*), 17
ugtm.ugtm_plot (*module*), 17
ugtm.ugtm_predictions (*module*), 17
ugtm.ugtm_preprocess (*module*), 18
ugtm.ugtm_sklearn (*module*), 16