PROVEE

Release 0.0.1

Michael Behrisch, Simen van Herpt, Dennis Owolabi, Sinie van de

Jul 04, 2021

CONTENTS:

CHAPTER

ONE

README

1.1 PROVEE - PROgressiVe Explainable Embeddings

1.2 Table of Contents

1.3 About

In this repository you will find PROVEE, short for Progressive Explainable Embeddings, a visual-interactive system for representing the embedding data spaces in a user-friendly 2D projection. The idea behind Progressive Analytics, such as described e.g. by Fekete and Primet, is to provide a rapid data exploration pipeline with a feedback loop from the system to the analyst with a latency below about 10 seconds. Research has shown that when performing exploratory analysis humans need a latency below about 10 seconds to remain focused and use their short-term memory efficiently. Therefore, PROVEE's goals are (1) to provide increasingly meaningful partial results as the algorithms execute and (2) provide visualizations that minimize distractions by not changing views excessively. All of this with a high scalability of the input data in combination with memory efficiency. *Note that these goals are adapted from the aforementioned publication.*

PROVEE's architecture includes (1) analysis algorithms (particularly, incremental projection algoritms like IPCA), (2) intuitive, local user interfaces/visualizations and (3) intermediate data storage and transfer. Core to our system is an innovative, progressive analysis workflow targeting a human-algorithm feedback-loop with a latency under ~10 seconds to maintain the user's efficiency during exploration tasks. PROVEE will be scalable to big data; generic (handle data from many application domains); and easy to use (requires no specialist programming from the user). Please also refer to our Performance and feature comparison to see the available (visualization and analysis) tools that we used as to compare PROVEE to.

1.4 Getting Started

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes. See *deployment* for notes on how to deploy the project on a live system.

1.4.1 Prerequisites

- Python 3
- Conda

1.4.2 Installing

Clone the latest Provee directory from Gitlab

```
git clone https://git.science.uu.nl/vig/provee/provee-local-projector.git
```

Create a conda environment. Your new environment will be named 'provee'.

```
conda env create -f environment.yml
```

Activate the environment.

```
conda activate provee
```

To deactivate the environment, use

```
conda deactivate
```

1.4.3 Running

To run the project, first activate the conda environment. Afterwards run `main.py` while in LocalProjector.

```
cd LocalProjector\
python main.py
```

1.5 Running the tests

The tests can be found under the folder 'LocalProjector/test'.

1.5.1 Basic Unit Tests

Tests can be run using pytest. First activate the conda environment. From the root folder, tests can be run using:

```
pytest LocalProjector/test/
```

To enable coverage, use:

```
pytest --cov=LocalProjector/src/ LocalProjector/test/
```

1.6 Usage

Notes about how to use the system are TBD, Video coming soon.

1.7 Deployment

If you want to deploy a live system refer to the Deployment Guide.

1.8 Built Using

- Vispy - 2D visualization
- Faiss - K-Nearest Neighbours
- gRPC - Microservices
- PyQt5 - Signaling & Service

1.9 Authors

- Michael Behrisch - Idea & Initial work
- Alex Telea - Idea & Initial work
- Dong Nguyen - Idea & Initial work
- Simen van Herpt - Backend & Infrastructure
- Dennis Owolabi - Code management & Infrastructure
- Sinie van der Ben - Comparison & Faiss

See also the list of contributors who participated in this project.

1.10 Acknowledgements

- Hat tip to anyone whose code was used

CHAPTER

TWO

SRC PACKAGE

2.1 Subpackages

2.1.1 src.KNN package

Submodules

src.KNN.faissKNN module

class src.KNN.faissKNN.**KNNService**
Bases: PyQt5.QtCore.QObject

The KNN microservice

Parameters **QObject** (*QObject*) – So that we can use QT signal/slots

calculateFinalVector(*sentence*)
Calculate the semantic request, for example king-man+woman

Parameters **sentence** (*string*) – the linalg expression (or single words)

Returns The calculated word vector

Return type resultedVector

checkWords(*wordList*, *wordDict*)
Check whether all words are found in the file in

Parameters

- **wordList** (*[string]*) – List of words used in the requested
- **wordDict** (*dict{string,[float]}*) – List of words found in the file with corresponding vectors

Returns Whether all words have been found

Return type Bool

clearVariables()
Clear all the variables to default (TODO, find more clean way of doing this)

getAllDistances(*request*, *context*)

getKNNBatch(*request*, *context*)
Gets the kNN for a batch of vectors.

Parameters

- **request** (*gRPC*) – standard gRPC (contains the messages)
- **request.vectors** (*[Vector]*) – the vectors of which the kNN must be returned
- **request.k** (*int*) – number of nearest neighbours that should be returned per vector
- **context** (*gRPC*) – standard gRPC

Returns

NeighboursBatch.neighbours contains Row objects with the nearest neighbours. This is in order of the vectors given.

Return type *NeighboursBatch*

getKNNRequest(*request*, *context*)
Implementation for gRPC message to connect to for the general requests

Parameters

- **request** (*gRPC*) – standard gRPC (contains the messages)
- **context** (*gRPC*) – standard gRPC

Returns gRPC message containing the words and distances

Return type grpc Neighbours objects

getProgress(*request*, *context*)
Implementation for the getProgress gRPC requset

Parameters

- **request** (*gRPC*) – standard gRPC (contains the messages)
- **context** (*gRPC*) – standard gRPC

Returns contains the integer of current progress

Return type grPC progressKNN

getVectors(*wordList*)
Find the vector per word in the given datafile

Parameters **wordList** (*[word]*) – List of used words

Returns The dictionary of word, wordVector combinations

Return type Dict

getWords(*indicesSorted*)
Searches and returns the words/labels of indices.

Parameters **indicesSorted** (*[int]*) – indices of vectors of which labels must be found in ascending order

Raises **Exception** – throws a exception when requested index was not found in file

Returns string}: contains the indices as key and requested labels as values

Return type {int

knnVector(*vector*, *k*)
Use the index object to extract closest words / distances and

Parameters

- **vector** (*[float]*) – The calculated word vector

- **k** (*int*) – How many neighbours we want to

Returns gRPC message containing the words and distances

Return type neighbours

printClosest(*indices*, *wordsDict*, *distances*)
For debugging, print the results of the KNN request

Parameters

- **indices** (*[[int]]*) – 2d list of the closest indices
- **distances** (*[[float]]*) – 2d list of distances to closest words (specified in indices)

progressSignal

startKNNService(*request*, *context*)
Setup the KNN serive

Parameters

- **request** (*gRPC*) – standard gRPC (contains the messages)
- **context** (*gRPC*) – standard gRPC

Returns returns void

Return type gRPC empty

stopKNNService(*request*, *context*)
Stop the current service

Parameters

- **request** (*gRPC*) – standard gRPC (contains the messages)
- **context** (*gRPC*) – standard gRPC

Returns grPC void

trainKNN()
Train the KNN with the given filePath

updatePercentage()
Recalculate the current percentage

src.KNN.faissKNN.**printMessage**(*message*)
Print the coloured message so that we can see in the output which microservice printed it

Parameters **message** (*string*) – The actual message

src.KNN.faissKNN.**serveServer**()
Setup the GRPC server

src.KNN.validationKNN module

class src.KNN.validationKNN.**ValidationKNN**
Bases: PyQt5.QtCore.QObject

The Validation KNN microservice

Parameters **QObject** (*QObject*) – So that we can use QT signal/slots

addTransformedPoints(*request*, *context*)
Add points from transformerService to buffer to validate later.

Parameters

- **request** (*gRPC*) – standard gRPC (contains the messages).
- **request.vectors** (*[vector]*) – contains protoKNN Vector object for each transformed point
- **context** (*gRPC*) – standard gRPC

Returns returns void

Return type gRPC empty

calcAccuracy()
Calculates the accuracy of each nearest neighbours comparison.

Returns float}: contains a amount of nearest neighbours as key and the corresponding accuracy as float (0-100%)

Return type {int

clearPoints(*request*, *context*)
Clear transformed points buffer so all previously added points are removed.

Parameters

- **request** (*gRPC*) – standard gRPC (contains the messages)
- **context** (*gRPC*) – standard gRPC

Returns returns void

Return type gRPC empty

clearVariables()
Clear all the variables to default

compareKNNBatch(*originalVectors*, *transformedVectors*, *nNeighbours*)
Compares the nearest neighbours of a batch of vectors with their full dimensial value and transformed value.

Parameters

- **originalVectors** (*[[float]]*) – List/array of full dimensional vectors
- **transformedVectors** (*[[float, float]]*) – List/array of transformed vectors
- **nNeighbours** (*int*) – Number of nearest neighbours that should be compared per vector (k in kNN).

Returns Amount of correct nearest neighbours found with amount of total nearest neighbours. (None, None) on failure.

Return type (int, int)

compareKNNs(*originalVectors*, *transformedVectors*)
Compares the kNNs of a batch of full dimensional vectors to transformed vectors. Compares for multiple k values.

Parameters

- **originalVectors** (*[[float]]*) – List/array of full dimensional vectors
- **transformedVectors** (*[[float, float]]*) – List/array of transformed vectors

Returns if comparisons were succesful

Return type bool

connectToOriginalKNN()
Connects to the kNN service to request the nearest neighbours of full dimensional vectors.

Returns if connecting was succesful

Return type bool

getAccuracy(*request*, *context*)
Gets the accuracy per nearest neighbour amount.

Parameters

- **request** (*gRPC*) – standard gRPC (contains the messages)
- **context** (*gRPC*) – standard gRPC

Returns contains a list of nearest neighbours and a list with the corresponding accuracy per nearest neighbour

Return type gRPC accuracyKNN

getOriginalNNBatch(*vectors*, *nNeighbours*)
Requests the kNN of a batch of full dimensional vectors from the kNN service.

Parameters

- **vectors** (*[[float]]*) – List/array of full dimensional vectors
- **nNeighbours** (*int*) – Number of nearest neighbours requested

Returns Per vector a list of nearest neighbours. Shape = (len(vectors), nNeighbours)

Return type np.array([[int]])

getProgress(*request*, *context*)
Implementation for the getProgress gRPC request

Parameters

- **request** (*gRPC*) – standard gRPC (contains the messages)
- **context** (*gRPC*) – standard gRPC

Returns contains the integer of current progress

Return type gRPC progressKNN

startKNNService(*request*, *context*)
Setup the KNN serive

Parameters

- **request** (*gRPC*) – standard gRPC (contains the messages)
- **context** (*gRPC*) – standard gRPC

Returns returns void

Return type gRPC empty

stopKNNService(*request*, *context*)
Stop the current service

Parameters

- **request** (*gRPC*) – standard gRPC (contains the messages)
- **context** (*gRPC*) – standard gRPC

Returns returns void

Return type gRPC empty

updateKNN()
Create a new kNN model on the transformed points.

Returns

Returns the transformed points used by the model. This prevents the ValidationKNN from getting out of sync when more points are added during updateKNN.

Return type [numpy.array([float, float])]

updatePercentage()
Recalculate the current percentage

validateProjection()
Calculate the accuracy of the transformed (projected) points compared to the original kNN with the full dimensions.

Returns float}: contains a amount of nearest neighbours as key and the corresponding accuracy as float (0-100%)

Return type {int

src.KNN.validationKNN.**printMessage**(*message*)
Print the coloured message so that we can see in the output which microservice printed it

Parameters **message** (*string*) – The actual message

src.KNN.validationKNN.**serveServer**()
Setup the GRPC server

Module contents

2.1.2 src.Projections package

Subpackages

src.Projections.CPP package

Module contents

src.Projections.Python package

Subpackages

src.Projections.Python.Projector package

Subpackages

src.Projections.Python.Projector.PCA package

Submodules

src.Projections.Python.Projector.PCA.PCAProjector module

class src.Projections.Python.Projector.PCA.PCAProjector.**PCA_Projector**
Bases: *src.Projections.Python.Projector.Projector.Projector*

PCA projector from sklearn

Parameters **Projector** (Projector) – General interface for projectors

clearVariables()
Clear all variables

getModel(*request*, *context*)
Return the model to gRPC message

Parameters

- **request** (*gRPC*) – standard gRPC (contains the messages)
- **context** (*gRPC*) – standard gRPC

Returns Return the byte model

Return type *model*

getPrecision()
Return the precision (Which sklearn PCA provides)

Returns [description]

Return type [type]

getProgress(*request*, *context*)
Implementation for the getProgress gRPC requset

Parameters

- **request** (*gRPC*) – standard gRPC (contains the messages)
- **context** (*gRPC*) – standard gRPC

Returns contains the integer of current progress

Return type grPC progressKNN

partialFit(*buff*)
Partial fit the buffer

Parameters **buff** (*[hdvector]*) – The list of the vectors

startProjectorService(*request*, *context*)
Set up the service

Parameters

- **request** (*gRPC*) – standard gRPC (contains the messages)

- **context** (*gRPC*) – standard gRPC

Returns gRPC void

stopProjectorService(*request*, *context*)
Stop the current service

Parameters

- **request** (*gRPC*) – standard gRPC (contains the messages)
- **context** (*gRPC*) – standard gRPC

Returns grPC void

trainPCA()
Starts training the projector

updatePercentage()
Recalculate the current percentage

src.Projections.Python.Projector.PCA.PCAProjector.**printMessage**(*message*)
Print the coloured message so that we can see in the output which microservice printed it

Parameters **message** (*string*) – The actual message

src.Projections.Python.Projector.PCA.PCAProjector.**serveServer**()
Setup the GRPC server

Module contents

src.Projections.Python.Projector.UMAP package

Submodules

src.Projections.Python.Projector.UMAP.UMAPProjector module

class src.Projections.Python.Projector.UMAP.UMAPProjector.**UMAPProjector**
Bases: *src.Projections.Python.Projector.Projector.Projector*

clearPointBufferExceptShared(*pointBuffer*)

clearVariables()
Clear all variables

formatAndCopyModel()

getModel(*request*, *context*)
Return the model to gRPC message

Parameters

- **request** (*gRPC*) – standard gRPC (contains the messages)
- **context** (*gRPC*) – standard gRPC

Returns Return the byte model

Return type *model*

getPrecision()

getProgress(*request*, *context*)
Implementation for the getProgress gRPC requset

Parameters

- **request** (*gRPC*) – standard gRPC (contains the messages)
- **context** (*gRPC*) – standard gRPC

Returns contains the integer of current progress

Return type grPC progressKNN

start(*self*, *priority: QThread.Priority = QThread.InheritPriority*)

startProjectorService(*request*, *context*)
Set up the service

Parameters

- **request** (*gRPC*) – standard gRPC (contains the messages)
- **context** (*gRPC*) – standard gRPC

Returns gRPC void

stopProjectorService(*request*, *context*)
Stop the current service

Parameters

- **request** (*gRPC*) – standard gRPC (contains the messages)
- **context** (*gRPC*) – standard gRPC

Returns grPC void

train()

updatePercentage()
Recalculate the current percentage

src.Projections.Python.Projector.UMAP.UMAPProjector.**printMessage**(*message*)
Print the coloured message so that we can see in the output which microservice printed it :param message: The actual message :type message: string

src.Projections.Python.Projector.UMAP.UMAPProjector.**serveServer**()
Setup the GRPC server

Module contents

Submodules

src.Projections.Python.Projector.Projector module

class src.Projections.Python.Projector.Projector.**Projector**
Bases: PyQt5.QtCore.QThread

Projector interface from which all Python projectors should inherit (WIP)

Parameters **QThread** (*QThread*) – So that we can use the QT signals / slots (Maybe change this QThread to Qobject)

clearVariables()
Clear all variables

getModel()

progressSignal

run(*self*)

stop()

train()

Module contents

src.Projections.Python.Transformer package

Submodules

src.Projections.Python.Transformer.ProjectionTransformer module

class src.Projections.Python.Transformer.ProjectionTransformer.**ProjectionTransformer**
Bases: *src.Projections.Python.Transformer.Transformer.Transformer*

The Transformer

Parameters **t** (Transformer) – General interface for transformers

checkIfAbort(*count*)
Check if the transformer should be aborter

Parameters **count** (*int*) – How many points have been processed

Returns Whether abort or not

Return type bool

clearVariables()
Clear all variables

getProgress(*request*, *context*)
Implementation for the getProgress gRPC requset

Parameters

- **request** (*gRPC*) – standard gRPC (contains the messages)
- **context** (*gRPC*) – standard gRPC

Returns contains the integer of current progress

Return type grPC progressKNN

startTransformerService(*request*, *context*)
Starts projecting the points

Parameters

- **request** (*gRPC*) – standard gRPC (contains the messages)
- **context** (*gRPC*) – standard gRPC

Yields *pointChunk* – gRPC messages containing the list of projected points

stopTransformerService(*request*, *context*)
Stop the current service

Parameters

- **request** (*gRPC*) – standard gRPC (contains the messages)
- **context** (*gRPC*) – standard gRPC

Returns grPC void

transformAndToGRPC(*buffer*)
Transform and create the gRPC messages

Parameters **buffer** (*[hdvector]*) – The list of the actual word vectors in

Returns list of points to be returned to main applications

Return type *pointChunk*

updatePercentage()
Recalculate the current percentage

src.Projections.Python.Transformer.ProjectionTransformer.**printMessage**(*message*)
Print the coloured message so that we can see in the output which microservice printed it

Parameters **message** (*string*) – The actual message

src.Projections.Python.Transformer.ProjectionTransformer.**serveServer**()
Setup the GRPC server

src.Projections.Python.Transformer.Transformer module

class src.Projections.Python.Transformer.Transformer.**Transformer**
Bases: PyQt5.QtCore.QThread

Transformer interface from which all Python transformers should inherit (WIP)

Parameters **QThread** (*QThread*) – So that we can use the QT signals / slots (Maybe change this QThread to Qobject)

newPointsSignal

progressSignal

run(*self*)

stop()

transform()

src.Projections.Python.Transformer.UMAPTransformer module

class src.Projections.Python.Transformer.UMAPTransformer.**UMAPTransformer**
Bases: *src.Projections.Python.Transformer.ProjectionTransformer.ProjectionTransformer*

The Transformer

Parameters **t** (Transformer) – General interface for transformers

alignTransformedSets(*transformed*, *prevTransformedSharedPoints*)

alignWithOrigin(*transformed*, *getSharedPoints*)

clearVariables()
Clear all variables

transformAndToGRPC(*pointBuffer*)
Transform and create the gRPC messages

Parameters **pointBuffer** (*[hdvector]*) – The list of the actual word vectors in

Returns list of points to be returned to main applications

Return type *pointChunk*

src.Projections.Python.Transformer.UMAPTransformer.**printMessage**(*message*)
Print the coloured message so that we can see in the output which microservice printed it

Parameters **message** (*string*) – The actual message

src.Projections.Python.Transformer.UMAPTransformer.**serveServer**()
Setup the GRPC server

Module contents

Module contents

Module contents

2.1.3 src.UI package

Submodules

src.UI.Renderer module

class src.UI.Renderer.**Renderer**(*pointRange*)
Bases: PyQt5.QtCore.QRunnable

Interface from which all renderes inherit

Parameters **QRunnable** (*QRunnable*) – To be able to use signal/slots (Maybe change this back to QObject since we use separate thread)

addPoints(*points*)

downloadDataset(*filePath*)

getWidget()

removeAllPoints()

update()

src.UI.Renderer2D module

class src.UI.Renderer2D.**Renderer2D**(*pointRange*)
Bases: *src.UI.Renderer.Renderer*

The visualiser which renderes all the points

Parameters **Renderer** (Renderer) – The visualier interface

addPoints(*points*)
Add a list of points to the visualier

Parameters **points** (*[points]*) – List of points

getWidget()
So that the provee object (main) can pass the widget to the UI

Returns The visualiser widget

Return type QWidget

removeAllPoints()
Remove all points (TODO We can probably use openGLWidget.clear())

setupGrids()
Creates the 3 grid axis for x,y,z

setupWidget()
Create the widget so that the UI can place it

update()
Every QTimer tick, update the new points, if there are any

src.UI.Renderer3D module

class src.UI.Renderer3D.**Renderer3D**(*pointRange*)
Bases: *src.UI.Renderer.Renderer*

The visualiser which renderes all the points

Parameters **Renderer** (Renderer) – The visualier interface

addPoints(*points*)
Add a list of points to the visualier

Parameters **points** (*[points]*) – List of points

downloadDataset(*filePath*)
Store the projetion on disk

getGrid(*rotation*, *translation*)
Get the grid GraphicsItem given color, sizes rotations etc.

Parameters

- **rotation** (*Tuple*) – The x,y,z rotatinos for the grid
- **translation** (*Tuple*) – The x,y,z translations for the grid

Returns The grid Item

Return type GLGridItem

getWidget()
So that the provee object (main) can pass the widget to the UI

Returns The visualiser widget

Return type QWidget

removeAllPoints()
Remove all points (TODO We can probably use openGLWidget.clear())

setupGrids()
Creates the 3 grid axis for x,y,z

setupWidget()
Create the widget so that the UI can place it

update()
Every QTimer tick, update the new points, if there are any

src.UI.SideGrip module

class src.UI.SideGrip.**SideGrip**(*parent*, *edge*)
Bases: PyQt5.QtWidgets.QWidget

mouseMoveEvent(*self*, *QMouseEvent*)

mousePressEvent(*self*, *QMouseEvent*)

mouseReleaseEvent(*self*, *QMouseEvent*)

resizeBottom(*delta*)

resizeLeft(*delta*)

resizeRight(*delta*)

resizeTop(*delta*)

src.UI.rangeslider module

class src.UI.rangeslider.**QRangeSlider**(*parent=None*)
Bases: PyQt5.QtWidgets.QWidget, src.UI.rangeslider.Ui_Form

drawValues()

end()

endValueChanged

getRange()

keyPressEvent(*self*, *QKeyEvent*)

max()

maxValueChanged

min()

minValueChanged

setBackgroundStyle(*style*)

setDrawValues(*draw*)

setEnd(*value*)

setMax(*value*)

setMin(*value*)

setRange(*start*, *end*)

setSpanStyle(*style*)

setStart(*value*)

start()

startValueChanged

src.UI.ui module

class src.UI.ui.**UI**(*main*, *parent=None*)

Bases: PyQt5.QtWidgets.QMainWindow, Ui_MainWindow

The actual UI widgets

Parameters

- **baseUIWidget** (*[type]*) – QT standard template
- **baseUIClass** (*[type]*) – QT standard template

Returns so that the Provee object (main) can use it

Return type The UI object

addModelToList(*name*)

addProjectors()

addWordsToTables(*wordTuples*)

clearWordTable()

clickedMainButton()
Disable/enable the left menu

closeWindow()
Close the window

property gripSize

handleConnections()
Handle the toolbar connections (Maybe transport this to main.py)

maximizeWindow()
Maximize the window

minimizeWindow()
Minimize the window

mouseMoveEvent(*self*, *QMouseEvent*)

mousePressEvent(*self*, *QMouseEvent*)

progressBarValue(*value*, *widget*, *color*)
Calculate the angle of the circular loading and set it

Parameters

- **value** (*int*) – New Percentage
- **widget** (*QWidget*) – The widget of the progress Bar
- **color** (*string*) – The used color

resizeEvent(*self*, *QResizeEvent*)

setGripSize(*size*)

setValue(*labelPercentage*, *progressBarName*, *color*, *value*)
Set the value of a circular progress bar (KNN, Projector or Transformer)

Parameters

- **labelPercentage** (*int*) – New Percentage
- **progressBarName** (*QWidget*) – The widget of the progress Bar
- **color** (*string*) – The used color
- **value** (*int*) – The new progress percentage

setVisualiser(*widget*)
Put the visualiser in the main stackedwidget, and select it from

Parameters **widget** (*QWidget*) – The visualier

setupPercentageLoaders()
Set up the circular percentage loaders

setupWordTable()

swapSettingsScreen()
Being able to switch the main screen (between visualiser and config widget (WIP))

updateFittingPercentage(*percentage*)
Update the Projector progress percentage

Parameters **percentage** (*int*) – percentage

updateGrips()

updateKNNPercentage(*percentage*)
Update the KNN progress percentage

Parameters **percentage** (*int*) – percentage

updateTransformPercentage(*percentage*)
Update the Transformer progress percentage

Parameters **percentage** (*int*) – percentage

Module contents

2.1.4 src.generated package

Subpackages

src.generated.Python package

Submodules

src.generated.Python.knn_pb2 module

Generated protocol buffer code.

class src.generated.Python.knn_pb2.**Neighbours**
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

rows
Field provee.Neighbours.rows

class src.generated.Python.knn_pb2.**NeighboursBatch**
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

neighbours
Field provee.NeighboursBatch.neighbours

class src.generated.Python.knn_pb2.**Row**
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

distance
Field provee.Row.distance

word
Field provee.Row.word

wordIndex
Field provee.Row.wordIndex

class src.generated.Python.knn_pb2.**Vector**
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

point
Field provee.Vector.point

class src.generated.Python.knn_pb2.**accuracyKNN**
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

accuracy
Field provee.accuracyKNN.accuracy

nNeighbours
Field provee.accuracyKNN.nNeighbours

class src.generated.Python.knn_pb2.**addTransformedPointsRequest**
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

vectors
Field provee.addTransformedPointsRequest.vectors

class src.generated.Python.knn_pb2.**allDistanceKNNRequest**
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

word
Field provee.allDistanceKNNRequest.word

class src.generated.Python.knn_pb2.**distanceList**
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

distances
Field provee.distanceList.distances

indices
Field provee.distanceList.indices

class src.generated.Python.knn_pb2.**knnBatchRequest**
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

k
Field provee.knnBatchRequest.k

vectors
Field provee.knnBatchRequest.vectors

class src.generated.Python.knn_pb2.**knnRequest**
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

k
Field provee.knnRequest.k

words
Field provee.knnRequest.words

class src.generated.Python.knn_pb2.**progressKNN**
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

progress
Field provee.progressKNN.progress

class src.generated.Python.knn_pb2.**startKNN**
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

path
Field provee.startKNN.path

src.generated.Python.knn_pb2_grpc module

Client and server classes corresponding to protobuf-defined services.

class src.generated.Python.knn_pb2_grpc.**KNN**
Bases: object

Interface exported by the server.

static getAllDistances(*request*, *target*, *options=()*, *channel_credentials=None*, *call_credentials=None*, *insecure=False*, *compression=None*, *wait_for_ready=None*, *timeout=None*, *metadata=None*)

static getKNNBatch(*request*, *target*, *options=()*, *channel_credentials=None*, *call_credentials=None*, *insecure=False*, *compression=None*, *wait_for_ready=None*, *timeout=None*, *metadata=None*)

static getKNNRequest(*request*, *target*, *options=()*, *channel_credentials=None*, *call_credentials=None*, *insecure=False*, *compression=None*, *wait_for_ready=None*, *timeout=None*, *metadata=None*)

static getProgress(*request*, *target*, *options=()*, *channel_credentials=None*, *call_credentials=None*, *insecure=False*, *compression=None*, *wait_for_ready=None*, *timeout=None*, *metadata=None*)

static startKNNService(*request*, *target*, *options=()*, *channel_credentials=None*, *call_credentials=None*, *insecure=False*, *compression=None*, *wait_for_ready=None*, *timeout=None*, *metadata=None*)

static stopKNNService(*request*, *target*, *options=()*, *channel_credentials=None*, *call_credentials=None*, *insecure=False*, *compression=None*, *wait_for_ready=None*, *timeout=None*, *metadata=None*)

class src.generated.Python.knn_pb2_grpc.**KNNServicer**
Bases: object

Interface exported by the server.

getAllDistances(*request*, *context*)
Missing associated documentation comment in .proto file.

getKNNBatch(*request*, *context*)
Missing associated documentation comment in .proto file.

getKNNRequest(*request*, *context*)
Missing associated documentation comment in .proto file.

getProgress(*request*, *context*)
Missing associated documentation comment in .proto file.

startKNNService(*request*, *context*)
Missing associated documentation comment in .proto file.

stopKNNService(*request*, *context*)
Missing associated documentation comment in .proto file.

class src.generated.Python.knn_pb2_grpc.**KNNStub**(*channel*)
Bases: object

Interface exported by the server.

class src.generated.Python.knn_pb2_grpc.**ValidationKNN**
Bases: object

Missing associated documentation comment in .proto file.

static addTransformedPoints(*request*, *target*, *options=()*, *channel_credentials=None*, *call_credentials=None*, *insecure=False*, *compression=None*, *wait_for_ready=None*, *timeout=None*, *metadata=None*)

static clearPoints(*request*, *target*, *options=()*, *channel_credentials=None*, *call_credentials=None*, *insecure=False*, *compression=None*, *wait_for_ready=None*, *timeout=None*, *metadata=None*)

static getAccuracy(*request*, *target*, *options=()*, *channel_credentials=None*, *call_credentials=None*, *insecure=False*, *compression=None*, *wait_for_ready=None*, *timeout=None*, *metadata=None*)

static getProgress(*request*, *target*, *options=()*, *channel_credentials=None*, *call_credentials=None*, *insecure=False*, *compression=None*, *wait_for_ready=None*, *timeout=None*, *metadata=None*)

static startKNNService(*request*, *target*, *options=()*, *channel_credentials=None*, *call_credentials=None*, *insecure=False*, *compression=None*, *wait_for_ready=None*, *timeout=None*, *metadata=None*)

static stopKNNService(*request*, *target*, *options=()*, *channel_credentials=None*, *call_credentials=None*, *insecure=False*, *compression=None*, *wait_for_ready=None*, *timeout=None*, *metadata=None*)

class src.generated.Python.knn_pb2_grpc.**ValidationKNNServicer**
Bases: object

Missing associated documentation comment in .proto file.

addTransformedPoints(*request*, *context*)
Missing associated documentation comment in .proto file.

clearPoints(*request*, *context*)
Missing associated documentation comment in .proto file.

getAccuracy(*request*, *context*)
Missing associated documentation comment in .proto file.

getProgress(*request*, *context*)
Missing associated documentation comment in .proto file.

startKNNService(*request*, *context*)
Missing associated documentation comment in .proto file.

stopKNNService(*request*, *context*)
Missing associated documentation comment in .proto file.

class src.generated.Python.knn_pb2_grpc.**ValidationKNNStub**(*channel*)
Bases: object

Missing associated documentation comment in .proto file.

src.generated.Python.knn_pb2_grpc.**add_KNNServicer_to_server**(*servicer*, *server*)

src.generated.Python.knn_pb2_grpc.**add_ValidationKNNServicer_to_server**(*servicer*, *server*)

src.generated.Python.projector_pb2 module

Generated protocol buffer code.

class src.generated.Python.projector_pb2.**model**
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

model
Field provee.model.model

class src.generated.Python.projector_pb2.**progressProjector**
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

progress
Field provee.progressProjector.progress

class src.generated.Python.projector_pb2.**startProjector**
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

path
Field provee.startProjector.path

src.generated.Python.projector_pb2_grpc module

Client and server classes corresponding to protobuf-defined services.

class src.generated.Python.projector_pb2_grpc.**Projector**
Bases: object

Interface exported by the server.

static getModel(*request*, *target*, *options=()*, *channel_credentials=None*, *call_credentials=None*, *insecure=False*, *compression=None*, *wait_for_ready=None*, *timeout=None*, *metadata=None*)

static getProgress(*request*, *target*, *options=()*, *channel_credentials=None*, *call_credentials=None*, *insecure=False*, *compression=None*, *wait_for_ready=None*, *timeout=None*, *metadata=None*)

static startProjectorService(*request*, *target*, *options=()*, *channel_credentials=None*, *call_credentials=None*, *insecure=False*, *compression=None*, *wait_for_ready=None*, *timeout=None*, *metadata=None*)

static stopProjectorService(*request*, *target*, *options=()*, *channel_credentials=None*, *call_credentials=None*, *insecure=False*, *compression=None*, *wait_for_ready=None*, *timeout=None*, *metadata=None*)

class src.generated.Python.projector_pb2_grpc.**ProjectorServicer**
Bases: object

Interface exported by the server.

getModel(*request*, *context*)
Missing associated documentation comment in .proto file.

getProgress(*request*, *context*)
Missing associated documentation comment in .proto file.

startProjectorService(*request*, *context*)
Missing associated documentation comment in .proto file.

stopProjectorService(*request*, *context*)
Missing associated documentation comment in .proto file.

class src.generated.Python.projector_pb2_grpc.**ProjectorStub**(*channel*)
Bases: object

Interface exported by the server.

src.generated.Python.projector_pb2_grpc.**add_ProjectorServicer_to_server**(*servicer*, *server*)

src.generated.Python.transformer_pb2 module

Generated protocol buffer code.

class src.generated.Python.transformer_pb2.**point**
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

clusterID
Field provee.point.clusterID

id
Field provee.point.id

x
Field provee.point.x

y
Field provee.point.y

class src.generated.Python.transformer_pb2.**pointChunk**
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

points
Field provee.pointChunk.points

class src.generated.Python.transformer_pb2.**progressTransformer**
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

progress
Field provee.progressTransformer.progress

class src.generated.Python.transformer_pb2.**startTransformer**
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

model
Field provee.startTransformer.model

path
Field provee.startTransformer.path

src.generated.Python.transformer_pb2_grpc module

Client and server classes corresponding to protobuf-defined services.

class src.generated.Python.transformer_pb2_grpc.**Transformer**
Bases: object

Interface exported by the server.

static getProgress(*request*, *target*, *options=()*, *channel_credentials=None*, *call_credentials=None*, *insecure=False*, *compression=None*, *wait_for_ready=None*, *timeout=None*, *metadata=None*)

static startTransformerService(*request*, *target*, *options=()*, *channel_credentials=None*, *call_credentials=None*, *insecure=False*, *compression=None*, *wait_for_ready=None*, *timeout=None*, *metadata=None*)

static stopTransformerService(*request*, *target*, *options=()*, *channel_credentials=None*, *call_credentials=None*, *insecure=False*, *compression=None*, *wait_for_ready=None*, *timeout=None*, *metadata=None*)

class src.generated.Python.transformer_pb2_grpc.**TransformerServicer**
Bases: object

Interface exported by the server.

getProgress(*request*, *context*)
Missing associated documentation comment in .proto file.

startTransformerService(*request*, *context*)
Missing associated documentation comment in .proto file.

stopTransformerService(*request*, *context*)
Missing associated documentation comment in .proto file.

class src.generated.Python.transformer_pb2_grpc.**TransformerStub**(*channel*)
Bases: object

Interface exported by the server.

src.generated.Python.transformer_pb2_grpc.**add_TransformerServicer_to_server**(*servicer*, *server*)

Module contents

Module contents

2.1.5 src.utils package

Submodules

src.utils.helperFunctions module

helperFunctions.py contains utility functions that could be used anywhere in the project.

src.utils.helperFunctions.**distance**(*p0*, *p1*)

src.utils.helperFunctions.**lineToVector**(*line*)
Parses a line to an object with a vector.

Parameters **line** (*str*) – The line that needs to be parsed, must contain a object followed by a numbers

Returns The object and its vector

Return type (str, [float])

src.utils.helperFunctions.**printColoured**(*message*, *service*, *color*)
Print the message in the given color (should be different for each microservice)

Parameters

- **message** (*string*) – The actual message
- **service** (*string*) – The name of the microservice
- **color** (*string*) – The name of the color used

Module contents

2.2 Submodules

2.2.1 src.constants module

Contains all the constant values, TODO: use a config file for this for user individual settings

2.2.2 src.serviceManager module

class src.serviceManager.**ServiceManager**
Bases: PyQt5.QtCore.QObject

Handles the connections to the microservices (KNN,Projector,Transformer)

Parameters **QObject** (*QObject*) – So that we can use signals/slots

KNNprogressSignal

ProjectorprogressSignal

TransformerprogressSignal

addPointsToValidationKNN(*points*)
Add transformed points to the validation kNN to validate precision when later requested.

Parameters **points** (*[[float]]*) – transformed points in 2D/3D

changeProjector(*name*)
Change the active projector and transformer types.

Parameters **name** (*string*) – name of projection method

Returns returns true if projector type was changed (false if already correct)

Return type bool

clearValidationKNN()
Clear validation kNN buffer with all transformed points of previous transformation.

densityRequestSignal

getKNNProgress()
Send a gRPC message to KNN to get the progress

getKNNRequest(*sentence*)
Send a gRPC message to KNN to get the result of KNN request

Parameters **sentence** (*string*) – The linalg expression of the request. For example, king - man + woman

getPoints(*filePath*, *m*)
Slot for the getPoints timer, retrieves the points from the transformer

Parameters

- **filePath** (*string*) – The path to the file we want to transform (note doesnt have to be the same one as the one which the model was projected on (WIP))
- **m** (*bytes*) – the pickle result for python of the model

getProjectorModel()
Send a gRPC message to Projector to get the current trained model

getProjectorProgress()
Send a gRPC message to projector to get the progress

getServiceProgress(*serviceType*, *name*, *serviceProtos*, *signal*, *timer*)
Send a gRPC message to a service to get the progress

getTransformerProgress()
Send a gRPC message to transformer to get the progress

isServiceActive(*serviceType*)
Gives the current active service of a service type.

Parameters **serviceType** (*string*) – service type as defined as keys in self.stubs

Returns the stub of the requested service

Return type gRPC stub

knnRequestSignal

makeDensityRequest(*densityWord*)

newPointsSignal

newProjectorModelSignal

setupKNNs()
Setup kNN microservices.

setupProjectors()
Setup projector microservices.

setupService(*serviceType*, *name*, *scriptPath*, *channelAddress*, *stubFunc*, *serviceProtos*)
Create a separate process for the projector (gRPC)

Parameters **filePath** (*string*) – The path to the file containing the data embedding (which we want to project)

setupTransformers()
Start up the transformer microservices.

startKNN(*filePath*)
Actually start the KNN service for a file

Parameters **filePath** (*string*) – Path to the file with embeddings

startProjection(*filePath*)
Starts the projection on the active projector service.

Parameters **filePath** (*string*) – The path to the file containing the data embedding (which we want to project)

startTransformer(*filePath*, *m*)
Gets the projection (gRPC)

Parameters

- **filePath** (*string*) – The path to the file we want to transform (note doesnt have to be the same one as the one which the model was projected on (WIP))
- **m** (*bytes*) – the pickle result for python of the model

startValidateProjection(*filePath*)
Starts validation kNN service to validate the projection.

Parameters **filePath** (*string*) – Path to the file with embeddings

stopTransformer()
Stop the projection

2.3 Module contents

CHAPTER

THREE

CODE OF CONDUCTS

The goal is to maintain a diverse community that's pleasant for everyone. That's why we would greatly appreciate it if everyone contributing to and interacting with the community also followed this Code of Conduct.

The Code of Conduct covers our behavior as members of the community, in any forum, mailing list, wiki, website, Internet relay chat (IRC), public meeting or private correspondence.

Our Code of Conduct is heavily based on the Celery Code of Conduct.

3.1 Be considerate

Your work will be used by other people, and you in turn will depend on the work of others. Any decision you take will affect users and colleagues, and we expect you to take those consequences into account when making decisions. Even if it's not obvious at the time, our contributions to PROVEE will impact the work of others. For example, changes to code, infrastructure, policy, documentation and translations during a release may negatively impact others' work.

3.2 Be respectful

The PROVEE community and its members treat one another with respect. Everyone can make a valuable contribution to PROVEE. We may not always agree, but disagreement is no excuse for poor behavior and poor manners. We might all experience some frustration now and then, but we cannot allow that frustration to turn into a personal attack. It's important to remember that a community where people feel uncomfortable or threatened isn't a productive one. We expect members of the PROVEE community to be respectful when dealing with other contributors as well as with people outside the PROVEE project and with users of PROVEE.

3.3 Be collaborative

Collaboration is central to PROVEE and to the larger free software community. We should always be open to collaboration. Your work should be done transparently and patches from PROVEE should be given back to the community when they're made, not just when the distribution releases. If you wish to work on new code for existing upstream projects, at least keep those projects informed of your ideas and progress. It many not be possible to get consensus from upstream, or even from your colleagues about the correct implementation for an idea, so don't feel obliged to have that agreement before you begin, but at least keep the outside world informed of your work, and publish your work in a way that allows outsiders to test, discuss, and contribute to your efforts.

3.4 When you disagree, consult others

Disagreements, both political and technical, happen all the time and the PROVEE community is no exception. It's important that we resolve disagreements and differing views constructively and with the help of the community and community process. If you really want to go a different way, then we encourage you to make a derivative distribution or alternate set of packages that still build on the work we've done to utilize as common of a core as possible.

3.5 When you're unsure, ask for help

Nobody knows everything, and nobody is expected to be perfect. Asking questions avoids many problems down the road, and so questions are encouraged. Those who are asked questions should be responsive and helpful. However, when asking a question, care must be taken to do so in an appropriate forum.

3.6 Step down considerately

Developers on every project come and go and PROVEE is no different. When you leave or disengage from the project, in whole or in part, we ask that you do so in a way that minimizes disruption to the project. This means you should tell people you're leaving and take the proper steps to ensure that others can pick up where you left off.

CHAPTER

FOUR

COMPARISON

As mentioned before, PROVEE is a user-friendly tool for 2D high scalability embedding projections, but it is not the only tool that aims to visualize embeddings. To build the optimal embedding projector we have looked at existing tools and examined what worked well using these tools. But most importantly: what was missing? Other's mistakes can be a fruitful sources of information. Mistakes can only imPROVEE our library!

There are several libraries and tools that try to visualize embeddings or parts of embeddings, but only a few have visualization of embeddings as the main purpose. The most important libraries/tools we have looked at to create PROVEE:

- Vec2graph
- Tensorflow Embedding Projector
- Whatlies
- Parallax

Libraries and tools were examined for their scalability, user-friendliness, responsiveness and advantages and disadvantages. We were also interested in their back-end and data storage and transfer.

4.1 Vec2graph

Vec2graph is a library by Katricheva et al.(2020) for visualizing word embeddings as graphs. The 2D graph is created based on cosine similarity between points and between neighbours. Edges between the nodes display semantic similarity between the embeddings. Graphs can contain nodes that link to other graphs and are displayed using an .html file. Vec2graph shines in its ease of use as graphs can be displayed using 2 or 3 lines of code, but the projections are not suited to display many data points at once.

4.2 Whatlies

Whatlies is a project developed by Rasa. The goal of the project is to create an API that supports many languages, such as SpaCy, Gensim and FastText to display word or sentence embeddings. Whatlies enables users to easily display data in interactive 2D graphs using Jupyter Notebooks. The axes can be defined by dimension reduction methods, such as PCA or UMAP, but also special queries using vector arithmetics. For instance, 'king - man' can be the y-axis and 'woman' can be the x-axis. The special thing about Whatlies is the support for vector arithmetic on embeddings, which can be visualized directly in the interactive plots (they are inspired by Parallax!). It has a high scalability of the input, but the overall goal is to visualize smaller groups of words. It is not able to display more than 5000 embeddings as well. Furthermore, the tool requires little programming knowledge and has a clear documentation on their Github.io page, including examples on how to use the tool.

4.3 Tensorflow Embedding Projector

The Embedding Projector is part of the Tensorboard. It can graphically represent embeddings in a 2D or 3D space. These embeddings can be anything, as long as they can be converted to a tab separated file. The tool has a high scalability and is suited to display many data points at once. It is possible to use the Embedding Projector locally or directly using the web-browser. The user can interactively explore the embedding space, varying many parameters, easily switching from PCA to UMAP, 2D to 3D. A disadvantage of the tool is the limitation of dimensionality reduction methods as preprocessing method to display the embeddings.

4.4 Parallax

Last but not least, there is Parallax. Parallax is a tool to display word embedding spaces, suited for many embeddings with high dimensions. The tool is suited to display many data points at once. Most interesting is the idea on vector arithmetics as axes in the article accompanying the tool of Molino et al. (2019). The axes of their tool can be obtained using PCA or t-SNE, but they propose a special approach for the specification of the axes: axes that are the result of vector arithmetics formulas on these embeddings. This results in axes such as the average of two words or the most frequently occuring word in the data set. The major drawback of the tool is the slow loading and reaction time when using >10.000 points. Another difficulty of the tool is the required programming exprience. Some knowledge of programming is required.

4.5 PROVEE

Our current project PROVEE tries to overcome all the major drawbacks of the studied tools, by designing a user-friendly, responsive tool, suitable for many embeddings with high dimensionality. The highest priority of our tool is providing a high scalability of the projection: thousands of points with multidimensional embeddings. The embeddings can be anything. Not only word embeddings, but image embeddings, DNA embeddings, sentence embeddigns, you name it. Other priorities are low memory footprint and ease of use: programming experience is not required, which makes the tool easy to use.

4.6 Overview

In this section we would like to give an overview of the tools, including characteristics we think of as important in developing PROVEE.

	Vec2graph	Tensorflow Embedding Projector	Whatlies	Parallax	PROVEE
2D interactive graph					
PCA, t-SNE, UMAP					
Vector Arithmetics axes					
Easy to use					
Display > 100.000 points					
Other than word embeddings					
Load multiple data sets					

4.6.1 Analytical Process: when to use which tool?

Context Differences

Words can have different meanings in different contexts. As a result, the embeddings of the same words in different contexts are different. It is possible to investigate these differences. PROVEE has the option to load in different data sets. Using the same words on the axes, but from different embedding sets, can display the meanings of similar words in diverse contexts.

Molino et al. (2019) display that Parallax is able to perform a similar task, using the co-occurence matrix. Whatlies and the Embedding Projector are likely to perform the task as well, but no tool is able to load in two data sets separately.

Local Embedding Inspection

Although the high scalability of projection is the main drive of PROVEE, the tool is suited to explore local embedding neighbourhoods. PROVEE displays embeddings in an interactive 2D visualization, meaning that users are able to zoom in or out as they desire. Inspecting the local neighbourhood of one specific embedding is enabled by zooming into the embedding space.

Tools designed especially to display local neighbourhood are Vec2graph and Whatlies. These tools have a low scalability of projection, because they aim to visualize the direct neighbourhood of a point. PROVEE is capable of providing such a local view, but this is not the direct goal of the tool. The Embedding Projector and Parallax are constructed in a similar way: not designed to display local neighbourhoods directly, but it is possible using these tools.

Bias Detection

A full set of embeddings can be displayed in 2D using specific axes to detect bias. The most common example is to put 'man' and 'woman' on the x- and y-axis for the detection of gender bias. Word embeddings are placed in the embedding space, relating their dimensions to the word 'man' and 'woman', as gender bias can be found by a the direction in the word embedding Bolukbasi et al. (2016). Words that are biased towards 'man' lie below the diagonal of the embedding space; words that are biased towards 'woman' lie above this diagonal.

Not only PROVEE is suited for bias detection. One could also use Parallax, Whatlies or the Tensorflow Embedding Projector. PROVEE is able to display more points than Whatlies and display these points faster than Parallax, which makes PROVEE suitable for bias detection in a data set containing many embeddings.

Analogy Task

The analogy task can be seen as the similarity between pairs of words. Already clear from the fact that bias can be detected in word embeddings, Mikolov et al. (2013) showed that embeddings contain linguistic regularities and patterns. These patterns become visible in the example 'Madrid' - 'Spain' + 'France', which is very close to 'Paris' (shown by Mikolov et al. (2013)). We can use PROVEE for this analogy task. Placing 'Madrid' - 'Spain' on one axis and 'France' on the other, we can see which embeddings are suited for this analogy. .

Molino et al. (2019) provides an example in the paper of the implementation of the analogy task for using Parallax. As Whatlies supports these vector arithmetics, this tool is suited for the analogy task as well. A similar argument holds: PROVEE is able to display more points faster than Whatlies and Parallax.

4.7 Summary

A table summarizing the tasks and the suitable tools. It is possible to think of other tasks as well, such as investigating polysemous embeddings or contextual embeddings using BERT embeddings. We can see that Whatlies and Parallax can perform all tasks, but not in a similar way as PROVEE (less points, not able to load in two data sets etc.).

	Vec2graph	Tensorflow Embedding Projector	Whatlies	Parallax	PROVEE
Context differences					
Local embedding inspection					
Bias detection					
Analogy Task					

CHAPTER

FIVE

CONTRIBUTING

5.1 Introduction

First off, thank you for considering contributing to Provee! Most likely that means that you came across some limitations of other systems or you just want to play around with the coolest newest technology (like we do).

5.1.1 BUT why should I read this!??!

Our contribution guidelines helps to communicate that you respect the time of the developers managing and developing this open source project. In return, they should reciprocate that respect in addressing your issue, assessing changes, and helping you finalize your pull requests.

5.1.2 Which kind of contributions we are searching for.

PROVEE is an open source project with academic background. We love to receive contributions from our community — you! Generally, all contributions are more than welcome. Ideas for non-conventional contributions can be writing tutorials or blog posts, improving the documentation, submitting bug reports and feature requests or writing code which can be incorporated into Provee itself.

5.2 Code of Conducts

Please find our Code of Conduct here. We that this serious! Code of Conduct for more fun and happiness in PROVEE

5.3 Your First Contribution

> Unsure where to begin contributing to Atom? You can start by looking through `beginner` and `help-wanted` issues:

```
Beginner issues - issues which should only require a few lines of code, and a test or␣
↪two.
Help wanted issues - issues which should be a bit more involved than beginner issues.
```

Issue lists can be sorted by total number of comments. While not perfect, number of comments is a reasonable proxy for impact a given change will have.

5.3.1 Never contributed to open source before? No problemo

Here are a couple of friendly tutorials: http://makeapullrequest.com/ and http://www.firsttimersonly.com/ or use this *free* series, How to Contribute to an Open Source Project on GitHub.

At this point, you're ready to make your changes! Feel free to ask for help; everyone is a beginner at first :smile_cat: If a maintainer asks you to `rebase` your PR, they're saying that a lot of code has changed, and that you need to update your branch so it's easier to merge.

5.4 Getting started

For something that is bigger than a one or two line fix:

1. Create your own fork of the code
2. Do the changes in your fork
3. If you like the change and think the project could use it:
    - Be sure you have followed the code style for the project.
    - Note the Provee Code of Conduct.
    - Send a pull request.

Small contributions such as fixing spelling errors, where the content is small enough to not be considered intellectual property, can be submitted by a contributor as a patch. As a rule of thumb, changes are obvious fixes if they do not introduce any new functionality or creative thinking. As long as the change does not affect functionality, some likely examples include the following:

- Spelling / grammar fixes
- Typo correction, white space and formatting changes
- Comment clean up
- Bug fixes that change default return values or error codes stored in constants
- Adding logging messages or debugging output
- Changes to 'metadata' files like Gemfile, .gitignore, build scripts, etc.
- Moving source files from one directory or package to another

5.5 Code, commit message and labeling conventions

These sections should streamline the contributions you make to PROVEE.

5.5.1 Preferred style for code.

TDB

5.5.2 Commit message conventions.

We are using semantic versioning semver and particularly semantic-release in this project. Release Notes and Changelogs are automatically generated. It is important that you obey the Angular Commit Message Conventions

Each commit message consists of a mandatory header and optionally a body and/or a footer. The header has a special format that includes a type, a scope and a subject. Here `<type>(<scope>): <subject>` is the header:

```
<type>(<scope>): <subject>
<BLANK LINE>
<body>
<BLANK LINE>
<footer>
```

Header

Type must be one of the following:

- feat: A new feature
- fix: A bug fix
- docs: Documentation only changes
- style: Changes that do not affect the meaning of the code (white-space, formatting, missing semi-colons, etc)
- refactor: A code change that neither fixes a bug nor adds a feature
- perf: A code change that improves performance
- test: Adding missing or correcting existing tests
- ci: Changes to the GitLab Pipeline
- chore: Changes to the build process or auxiliary tools and libraries

Scope

The scope is optional. It is used to show which part of the project is affected. Therefore use it for a filename (without extenstion), a folder name to describe the part of the project changed. If a change affects a lot of parts in the project the scope should be left blank, e.g. `test: added missing unittests` affects the whole test folder so the scope is redundant.

Subject

Use the subject to give a brief summary of the changes. Do this in the present tense and start with a lower case, e.g. `feat(transformer): change transformer to support new projector types`. Note the usage of 'change' instead of 'changed' or 'changes'.

Body

The body can be used to further elaborate on the changes and the motivation behind it. For example:

```
fix(transformer): fix transformer crashing at end of transforming points

The transformer kept requesting points even when finished causing an error. The␣
↪transformer now first checks if all points have been handled.
```

Footer

The footer can be used to name breaking changes. Breacking changes are changes that cause backwards incompatibility. The footer should contain the changes and instructions to roll back if possible. Here is an example without a body:

```
perf(projector): remove projection type option, default set to PCA

BREAKING CHANGE: Support for projections other than PCA have been removed to improve␣
↪performance.
```

For more information see Angular Commit Message Conventions or for examples see AngularJS Git Commit Message Conventions.

Usage

Your code can be merged to master by making a merge request. Give the merge request the same title as your header. Enable the checkbox for `Squash commits`. Click on `Modify commit messages` and make sure that the **Squash commit message** follows the commit message convention. GitLab should automatically set the message to the merge request title which functions as header, but you can also add a body and footer, both having a new blank line above them. When your merge request is approved by an assignee and merged the pipeline will run semantic release and update the version and changelog according to your squashed commit message.

5.5.3 Labeling conventions for issues

We try to follow the StandardIssueLabels for open source projects

5.6 How to report a bug

5.6.1 Security FIRST!

Any security issues should be submitted directly to m.behrisch at uu.nl In order to determine whether you are dealing with a security issue, ask yourself these two questions:

- Can I access something that's not mine, or something I shouldn't have access to?
- Can I disable something for other people?

If the answer to either of those two questions are "yes", then you're probably dealing with a security issue. Note that even if you answer "no" to both questions, you may still be dealing with a security issue, so if you're unsure, just email us at m.behrisch at uu.nl.

5.6.2 How to file a bug report.

When filing an issue, make sure to answer these five questions:

1. What version of PROVEE are you using (VERSION.txt file)?
2. What operating system and browser are you using?
3. What did you do?
4. What did you expect to see?
5. What did you see instead? General questions should go to the provee mailing list instead of the issue tracker. The provee-community will directly answer or ask you to file an issue if you've tripped over a bug.

5.7 How to suggest a feature or enhancement

If you find yourself wishing for a feature that doesn't exist in PROVEE, you are probably not alone. There are bound to be others out there with similar needs. Many of the features that PROVEE has today have been added because our users saw the need. Open an issue on our issues list on GitLAB which describes the feature you would like to see, why you need it, and how it should work.

5.8 Code review process

The core team looks at Pull Requests on a regular basis in a weekly triage meeting that we hold in a public Teams meeting (link in the README under contributing). After feedback has been given we expect responses within two weeks. After two weeks we may close the pull request if it isn't showing any activity.

CHAPTER

SIX

DEPLOYMENT GUIDE

6.1 Table of Contents

6.2 Deployment Guide

empty for now

6.3 Hardware Requirements

empty for now

6.4 Software Requirements

empty for now

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

H

I

K

L

M

N

P

Q

R

S

X

Y