sourcepredict Documentation

Release 0.3.2

Maxime Borry

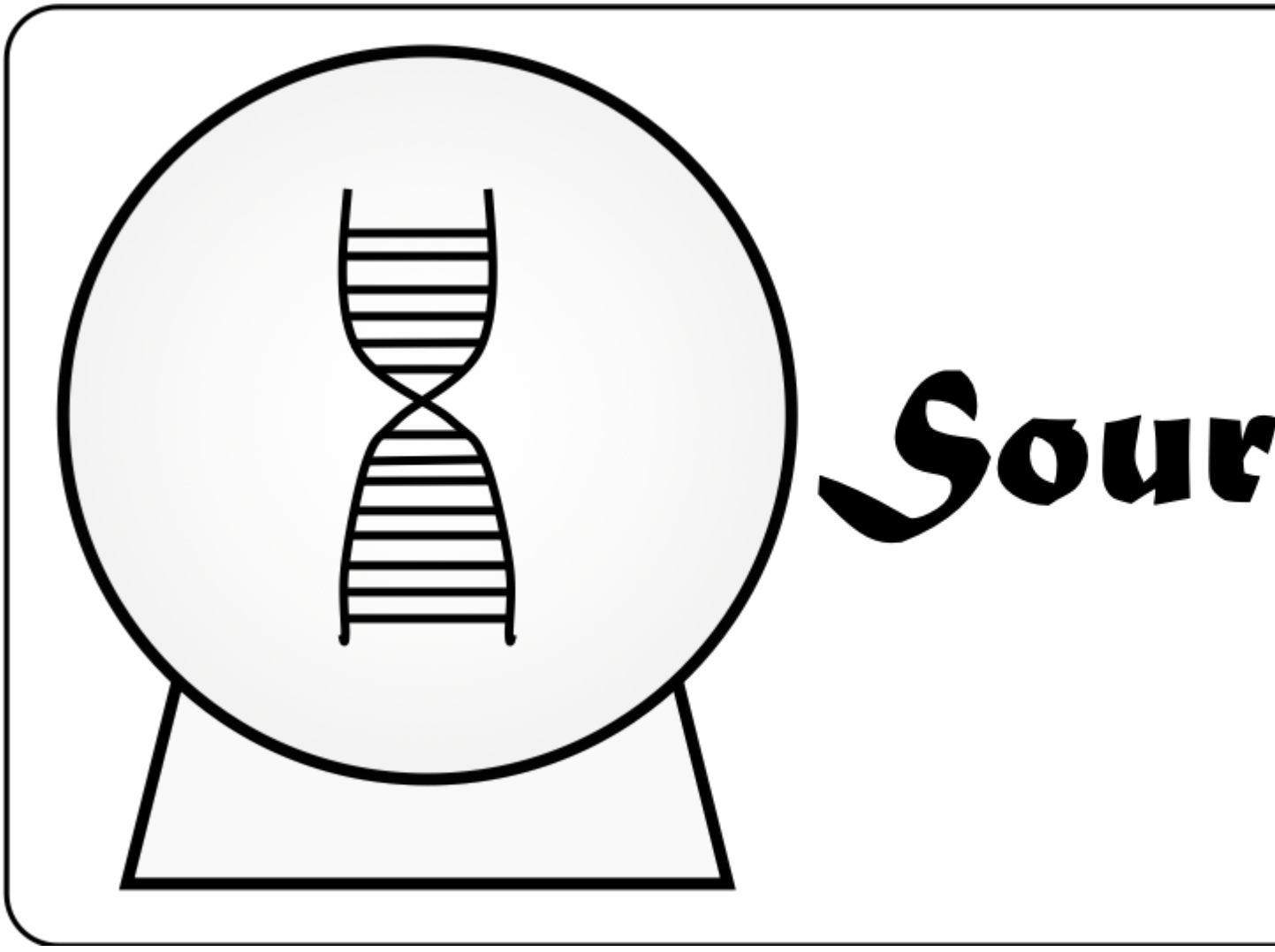
Aug 28, 2019

Contents:

Homepage: github.com/maxibor/sourcepredict

CHAPTER 1

Introduction



Prediction/source tracking of metagenomic samples source using machine learning

---

SourcePredict (github.com/maxibor/sourcepredict) is a Python package distributed through Conda, to classify and predict the origin of metagenomic samples, given a reference dataset of known origins, a problem also known as source tracking.DNA shotgun sequencing of human, animal, and environmental samples has opened up new doors to explore the diversity of life in these different environments, a field known as metagenomics.One aspect of metagenomics is investigating the community composition of organisms within a sequencing sample with tools known as taxonomic classifiers, such as Kraken.

In cases where the origin of a metagenomic sample, its source, is unknown, it is often part of the research question to predict and/or confirm the source. For example, in microbial archaelogy, it is sometimes necessary to rely on metagenomics to validate the source of paleofaeces. Using samples of known sources, a reference dataset can be established with the taxonomic composition of the samples, *i.e.* the organisms identified in the samples as features, and the sources of the samples as class labels. With this reference dataset, a machine learning algorithm can be trained to predict the source of unknown samples (sinks) from their taxonomic composition.Other tools used to perform the prediction of a sample source already exist, such as SourceTracker sourcetracker, which employs Gibbs sampling.However, the Sourcepredict results are easier interpreted since the samples are embedded in a human observable low-dimensional space. This embedding is performed by a dimension reduction algorithm followed by K-Nearest-Neighbours (KNN) classification.

CHAPTER 2

Installation

SourcePredict can be installed using the conda package manager:

```
$ conda install -c conda-forge -c etetoolkit -c bioconda -c maxibor
sourcepredict
```

CHAPTER 3

Usage

3.1 Running sourcepredict on the test dataset

```
$ wget https://raw.githubusercontent.com/maxibor/sourcepredict/master/data/test/dog_
↪test_sink_sample.csv -O dog_example.csv
$ wget https://raw.githubusercontent.com/maxibor/sourcepredict/master/data/modern_gut_
↪microbiomes_labels.csv -O sp_labels.csv
$ wget https://raw.githubusercontent.com/maxibor/sourcepredict/master/data/modern_gut_
↪microbiomes_sources.csv -O sp_sources.csv
$ sourcepredict -s sp_sources.csv -l sp_labels.csv dog_example.csv
```

3.2 Command line interface

```
$ sourcepredict -h
usage: SourcePredict v0.34 [-h] [-a ALPHA] [-s SOURCES] [-l LABELS]
                           [-n NORMALIZATION] [-dt DISTANCE] [-me METHOD]
                           [-kne NEIGHBORS] [-kw WEIGHTS] [-e EMBED] [-di DIM]
                           [-o OUTPUT] [-se SEED] [-k KFOLD] [-t THREADS]
                           sink_table

============================================================
SourcePredict v0.33
Coprolite source classification
Author: Maxime Borry
Contact: <borry[at]shh.mpg.de>
Homepage & Documentation: github.com/maxibor/sourcepredict
============================================================


positional arguments:
  sink_table   path to sink TAXID count table in csv format
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  -h, --help            show this help message and exit
  -a ALPHA              Proportion of sink sample in unknown. Default = 0.1
  -s SOURCES            Path to source csv file. Default =
                        data/modern_gut_microbiomes_sources.csv
  -l LABELS             Path to labels csv file. Default =
                        data/modern_gut_microbiomes_labels.csv
  -n NORMALIZATION      Normalization method (RLE | Subsample | GMPR | None).
                        Default = GMPR
  -dt DISTANCE          Distance method. (unweighted_unifrac | weighted_unifrac)
                        Default = weighted_unifrac
  -me METHOD            Embedding Method. TSNE, MDS, or UMAP. Default = TSNE
  -kne NEIGHBORS        Numbers of neigbors for KNN classication. Default = 0
                        (chosen by CV)
  -kw WEIGHTS           Sample weight function for KNN prediction (distance |
                        uniform). Default = distance.
  -e EMBED              Output embedding csv file. Default = None
  -di DIM               Number of dimensions to retain for dimension reduction.
                        Default = 2
  -o OUTPUT             Output file basename. Default =
                        <sample_basename>.sourcepredict.csv
  -se SEED              Seed for random generator. Default = 42
  -k KFOLD              Number of fold for K-fold cross validation in parameter
                        optimization. Default = 5
  -t THREADS            Number of threads for parallel processing. Default = 2
```

3.3 Command line arguments

3.3.1 sink_table

Sink TAXID count table in `csv` file format

Example sink count table file

TAXID	SINK_1	SINK_2
283	5	2
143	25	48

3.3.2 -alpha

Proportion of alpha of sink sample in unknown. Default = `0.1` $$\alpha \in [0,1]$$

Example:

```
-alpha 0.1
```

3.3.3 -s SOURCES

Path to source `csv` (training) file with samples in columns, and TAXIDs in rows. Default = `data/sourcepredict/modern_gut_microbiomes_sources.csv`

Example:

```
-s data/sourcepredict/modern_gut_microbiomes_sources.csv
```

Example source file :

TAXID	SAMPLE_1	SAMPLE_2
467	18	24
786	3	90

3.3.4 -l LABELS

Path to labels `csv` file of sources. Default = `data/modern_gut_microbiomes_labels.csv`

Example:

```
-l data/modern_gut_microbiomes_labels.csv
```

Example source file :

	labels
SAMPLE_1	Dog
SAMPLE_2	Human

3.3.5 -n NORMALIZATION

Normalization method. One of `RLE`, `CLR`, `Subsample`, or `GMPR`. Default = `GMPR`

3.3.6 -dt DISTANCE

Distance method. One of `unweighted_unifrac`, `weighted_unifrac`. Default = `weighted_unifrac`

Example:

```
-dt weighted_unifrac
```

3.3.7 -me METHOD

Embedding Method. One of `TSNE` or `UMAP`. Default = `TSNE`

Example:

```
-me TSNE
```

3.3.8 -kne NEIGHBORS

Numbers of neigbors for KNN classication. Default = 0 (chosen by CV).

Example:

```
-kne 30
```

> Setting the number of neighbors to 0 will let Sourcepredict choose the optimal number of neighbors for **classification**.However, for **source proportion estimation**, setting manually a higher number of samples (for example, 50) will help for better proportion estimations.See example 2 for illustration.

3.3.9 –kw WEIGHTS

Sample weight function for KNN prediction (distance | uniform). Default = distance.

Choose to give a uniform or distance based weights to neighbor samples in KNN algorithm.

> Distance base weights will work better for **classification** while uniform weigths will work better for **source proportion estimation**.See example 2 for illustration.

3.3.10 -e EMBED

File for saving embedding coordinates in `csv` format. Default = `None`

Example:

```
-e embed_coord.csv
```

3.3.11 -di DIM

Number of dimensions to retain for dimension reduction. Default = `2`

Example:

```
-di 2
```

3.3.12 -o OUTPUT

Sourcepredict Output file basename. Default = `<sample_basename>.sourcepredict.csv`

Example:

```
-o my_output
```

3.3.13 -se SEED

Seed for random number generator. Default = `42`

Example:

```
-se 42
```

3.3.14 -k KFOLD

Number of fold for K-fold cross validation in parameter optimization. Default = `5`

Example:

```
-k 5
```

3.3.15 -t THREADS

Number of threads for parallel processing. Default = `2`

Example:

```
-t 2
```

3.4 Choice of the taxonomic classifier

Different taxonomic classifiers will give different results, because of different algorithms, and different databases.

In order to produce correct results with Sourcepredict, **the taxonomic classifier and the database used to produce the *source* TAXID count table must be the same as the one used to produce the *sink* TAXID count table**.

Because Sourcepredict relies on machine learning, at least 10 samples per sources are required, but more source samples will lead to a better prediction by Sourcepredict.

Therefore, running all these samples through a taxonomic classifier ahead of Sourcepredict requires a non-negligeable computational time.

Hence the choice of the taxonomic classifier is a balance between precision, and computational time.

While this documentation doesn't intent to be a benchmark of taxonomic classifiers, the author of Sourcepredict has had decent results with Kraken2 and recommends it for its good compromise between precision and runtime.

The example *source* and *sink* data provided with Sourcepredict were generated with Kraken2.

> If you already have kraken report formatted results (from Kraken, KrakenUniq, Kraken2, Centrifuge, . . .), you can use the kraken_parse.py script to convert a kraken report to a column of a TAXID count table.

CHAPTER 4

Output

4.1 SourcePredict result file

File: `*.sourcepredict.csv`

This `csv` file contains the predicted proportion of each source in each sample. Like in any classification problem, the predicted source is the greatest proportion.

Example:

	ERR1915662
Canis_familiaris	0.9449678590674971
Homo_sapiens	0.027033026106258438
Soil	0.014110223165444446
unknown	0.013888891660799834

While in this example it is pretty clear that the `ERR1915662` sample is likely a dog, you may face situations where it will be less obvious. Looking at the embedding can therefore be useful to decide from which source(s) the sink sample is made up of.

4.2 Embedding csv file

This `csv` file contains the embedding of training in test samples in lower dimensions by TSNE or UMAP

Example:

	PC1	PC2	labels	name
SRR1175007	-28.858526	0.59231776	Homo_sapiens	SRR1175007
SRR042182	-22.14415	-0.47057405	Homo_sapiens	SRR042182
SRR061154	-30.210106	-2.0323594	Homo_sapiens	SRR061154
SRR061499	-25.546652	0.27987793	Homo_sapiens	SRR061499
SRR063469	-22.88011	1.1526666	Homo_sapiens	SRR063469
SRR062324	-25.50832	-0.25076494	Homo_sapiens	SRR062324
SRR1179037	-28.779644	0.1385772	Homo_sapiens	SRR1179037
SRR061236	-29.470839	-0.8973783	Homo_sapiens	SRR061236
SRR061456	-28.31991	-0.9834692	Homo_sapiens	SRR061456
SRR1761669	4.1411834	14.485897	Homo_sapiens	SRR1761669
SRR1761668	1.7706155	13.6566925	Homo_sapiens	SRR1761668
SRR1761675	3.2434833	16.020077	Homo_sapiens	SRR1761675
SRR3578625	24.127249	17.996181	Soil	SRR3578625
ERR1939165	28.738718	19.882471	Soil	ERR1939165
SRR3578645	24.138885	17.998867	Soil	SRR3578645
ERR1915662	-14.770308	-30.94284	sink	ERR1915662

See the example usage of Sourcepredict for a example of how to plot it.

CHAPTER 5

Methods

Starting with a numerical organism count matrix (samples as columns, organisms as rows, obtained by a taxonomic classifier) of merged references and sinks datasets, samples are first normalized relative to each other, to correct for uneven sequencing depth using the GMPR method (default). After normalization, Sourcepredict performs a two-step prediction algorithm. First, it predicts the proportion of unknown sources, *i.e.* which are not represented in the reference dataset. Second it predicts the proportion of each known source of the reference dataset in the sink samples.

Organisms are represented by their taxonomic identifiers (TAXID).

5.1 Prediction of unknown sources proportion

Let $S_i \in \{S_1, .., S_n\}$ be a sample from the normalized sinks dataset D_{sink}, ${o_j}^i \in \{{o_1}^i, .., {o_{n_o}}^i\}$ be an organism in S_i, and ${n_o}^i$ be the total number of organisms in S_i, with ${o_j}^i \in \mathbb{Z}+$.

Let m be the mean number of samples per class in the reference dataset, such that $m = \frac{1}{O}\sum_{i=1}^{O} S_i$.

For each S_i sample, I define $||m||$ estimated samples $U_k^{S_i} \in \{U_1^{S_i}, .., U_{||m||}^{S_i}\}$ to add to the reference dataset to account for the unknown source proportion in a test sample.

Separately for each S_i, a proportion denoted $\alpha \in [0, 1]$ (default = 0.1) of each of the ${o_j}^i$ organism of S_i is added to each $U_k^{S_i}$ samples such that $U_k^{S_i}({o_j}^i) = \alpha \cdot {x_i}_j$, where ${x_i}_j$ is sampled from a Gaussian distribution $\mathcal{N}\big(S_i({o_j}^i), 0.01\big)$.

The $||m||$ $U_k^{S_i}$ samples are then added to the reference dataset D_{ref}, and labeled as *unknown*, to create a new reference dataset denoted ${}^{unk}D_{ref}$.

To predict the proportion of unknown sources, a Bray-Curtis pairwise dissimilarity matrix of all S_i and $U_k^{S_i}$ samples is computed using scikit-bio. This distance matrix is then embedded in two dimensions (default) with the scikit-bio implementation of PCoA.

This sample embedding is divided into three subsets: ${}^{unk}D_{train}$ (64%), ${}^{unk}D_{test}$ (20%), and ${}^{unk}D_{validation}$(16%).

The scikit-learn implementation of KNN algorithm is then trained on ${}^{unk}D_{train}$, and the training accuracy is computed with ${}^{unk}D_{test}$.
This trained KNN model is then corrected for probability estimation of the unknown proportion using the scikit-learn implementation of Platt's scaling method with ${}^{unk}D_{validation}$.

The proportion of unknown sources in S_i, $p_u \in [0, 1]$ is then estimated using this trained and corrected KNN model.

Ultimately, this process is repeated independantly for each sink sample S_i of D_{sink}.

5.2 Prediction of known source proportion

First, only organism TAXIDs corresponding to the species taxonomic level are retained using the ETE toolkit. A weighted Unifrac (default) pairwise distance matrix is then computed on the merged and normalized training dataset D_{ref} and test dataset D_{sink} with scikit-bio.

This distance matrix is then embedded in two dimensions (default) using the scikit-learn implementation of t-SNE.

The 2-dimensional embedding is then split back to training ${}^{tsne}D_{ref}$ and testing dataset ${}^{tsne}D_{sink}$.

The training dataset ${}^{tsne}D_{ref}$ is further divided into three subsets: ${}^{tsne}D_{train}$ (64%), ${}^{tsne}D_{test}$ (20%), and ${}^{tsne}D_{validation}$ (16%).
The KNN algorithm is then trained on the train subset, with a five (default) cross validation to look for the optimum number of K-neighbors. The training accuracy is then computed with ${}^{tsne}D_{test}$. Finally, this second trained KNN model is also corrected for source proportion estimation using the scikit-learn implementation of the Platt's method with ${}^{tsne}D_{validation}$.

The proportion $p_{c_s} \in [0, 1]$ of each of the n_s sources $c_s \in \{c_1, .., c_{n_s}\}$ in each sample S_i is then estimated using this second trained and corrected KNN model.

5.3 Combining unknown and source proportion

Then for each sample S_i of the test dataset D_{sink}, the predicted unknown proportion p_u is then combined with the predicted proportion p_{c_s} for each of the n_s sources c_s of the training dataset such that $\sum_{c_s=1}^{n_s} s_c + p_u = 1$ where $s_c = p_{c_s} \cdot p_u$.

Finally, a summary table gathering the estimated sources proportions is returned as a `csv` file, as well as the t-SNE embedding sample coordinates.

CHAPTER 6

Custom sources

> Different taxonomic classifers will give different results, and **the taxonomic classifier used to produce the *source* TAXID count table must be the same as the one used to produce the *sink* TAXID count table**.

While there are many available taxonomic classifiers available to produce the source and sink TAXID table, the Sourcepredict author provide a simple pipeline to generate the source and sink TAXID table.

This pipeline is written using Nextflow, and handles the dependancies using conda. Briefly, this pipelines will firt trim and clip the sequencing files with AdapterRemoval before performing the taxonomic classification with Kraken2.

6.1 Pipeline installation

```
$ conda install -c bioconda nextflow
$ nextflow pull maxibor/kraken-nf
```

6.2 Running the pipeline

See the README of maxibor/kraken-nf

CHAPTER 7

Adding new sources to an existing source file

[1]:
```
import pandas as pd
import numpy as np
```

[24]:
```
def add_new_data(old_data, new_data, old_labels, label):
    """
    Update the sourcepredict learning table
    INPUT:
        old_data(str): path to csv file of existing sourcepredict source data table
        new_data(str): path to csv file of new TAXID count table, with TAXID as 1st column
        old_labels(str): path to sourcepredict csv file of labels
        label(str): scientific name of new sample's specie. Example: 'Sus_scrofa'
    OUTPUT:
        merged(pd.DataFrame): merged old and new source data table for sourcepredict
        labels(pd.DataFrame): updated labels data table
    """
    old = pd.read_csv(old_data, index_col=0)
    old = old.drop(['labels'], axis = 0)
    new = pd.read_csv(new_data)
    merged = pd.merge(left=old, right=new, how='outer', on='TAXID')
    merged = merged.fillna(0)
    old_labels = pd.read_csv(old_labels, index_col=0)
    new_labels = pd.DataFrame([label]*(new.shape[1]-1), new.columns[1:])
    new_labels.columns=['labels']
    labels = old_labels.append(new_labels)
    return(merged, labels)
```

[30]:
```
labs = add_new_data(old_data=old_data, new_data=new_data, old_labels=old_labels, label=label)[1]
```

[31]:
```
labs.to_csv("new_sources.csv")
```

CHAPTER 8

Sourcepredict example 1: Gut host species prediction

[1]:
```
import pandas as pd
import pandas_ml as pdml
```

In this example, we will use Sourcepredict and Sourcetracker2 applied to the example dataset provided in the Sourcepredict directory.

The example datasets contains the following samples: - *Homo sapiens* gut microbiome (1, 2, 3, 4, 5, 6) - *Canis familiaris* gut microbiome (1) - Soil microbiome (1, 2, 3)

8.1 Preparing the data

[2]:
```
cnt = pd.read_csv("../data/modern_gut_microbiomes_sources.csv", index_col=0)
labels = pd.read_csv("../data/modern_gut_microbiomes_labels.csv", index_col=0)
```

This is a TAXID count table containing the samples as columns headers, and the TAXID as row indices

[3]:
```
cnt.head()
```

[3]:
```
        SRR1175007   SRR042182   SRR061154   SRR061499  SRR063469   SRR062324  \
TAXID
0        3528337.0  11563613.0  10084261.0  20054993.0  8747525.0  12116517.0
6              0.0        78.0         0.0       127.0        0.0        79.0
7              0.0        78.0         0.0       127.0        0.0        79.0
9              0.0       129.0         0.0       153.0        0.0       151.0
10             0.0       160.0         0.0       193.0        0.0        99.0

        SRR1179037   SRR061236   SRR061456   SRR642021  ...  mgm4477903_3  \
TAXID                                                   ...
0        4191329.0  13992760.0  14825759.0  11083673.0  ...     6169203.0
6              0.0         0.0         0.0       172.0  ...          68.0
7              0.0         0.0         0.0       172.0  ...          68.0
9              0.0       165.0        96.0         0.0  ...           0.0
```

(continues on next page)

(continued from previous page)

```
10             0.0        55.0       249.0       238.0  ...            0.0

       mgm4477807_3  mgm4477874_3  mgm4477904_3  mgm4477804_3  mgm4477873_3  \
TAXID
0         8820851.0     5713837.0    10238500.0     5055930.0    10380594.0
6             247.0         211.0         156.0         147.0         383.0
7             247.0         211.0         156.0         147.0         383.0
9               0.0           0.0           0.0           0.0           0.0
10              0.0           0.0           0.0           0.0           0.0

       ERR1939166  SRR3578625  ERR1939165  SRR3578645
TAXID
0      13391896.0      1553.0  14802198.0       736.0
6          1353.0         0.0      1522.0         0.0
7          1353.0         0.0      1522.0         0.0
9            77.0         0.0        65.0         0.0
10          263.0         0.0       466.0         0.0

[5 rows x 432 columns]
```

The labels file contains the mapping of samples names with their actual origin (sources)

[4]:
```
labels.head()
```

[4]:
```
                labels
SRR1175007  Homo_sapiens
SRR042182   Homo_sapiens
SRR061154   Homo_sapiens
SRR061499   Homo_sapiens
SRR063469   Homo_sapiens
```

We will divide the source in training (95%) and testing (5%) dataset

[5]:
```
cnt_train = cnt.sample(frac=0.95, axis=1)
cnt_test = cnt.drop(cnt_train.columns, axis=1)
```

We also have to subset the labels file to only the training dataset

[6]:
```
train_labels = labels.loc[cnt_train.columns,:]
test_labels = labels.loc[cnt_test.columns,:]
```

8.2 Sourcepredict

Last but not least, we must export the files to `csv` to run sourcepredict

[7]:
```
cnt_train.to_csv("gut_species_sources.csv")
cnt_test.to_csv("gut_species_sinks.csv")
train_labels.to_csv("gut_species_labels.csv")
```

We'll now launch sourcepredict with the GMPR normalization method, and the t-SNE embedding, on 6 cores.

[8]:
```
%%time
!sourcepredict -s gut_species_sources.csv \
               -l gut_species_labels.csv \
```

(continues on next page)

(continued from previous page)

```
               -n GMPR \
               -m TSNE \
               -e example_embedding.csv \
               -t 6 gut_species_sinks.csv
```

```
Step 1: Checking for unknown proportion
  == Sample: SRR1175007 ==
        Adding unknown
        Normalizing (GMPR)
        Computing Bray-Curtis distance
        Performing MDS embedding in 2 dimensions
        KNN machine learning
        Training KNN classifier on 6 cores...
        -> Testing Accuracy: 1.0
        ---------------
        - Sample: SRR1175007
                  known:98.68%
                  unknown:1.32%
  == Sample: SRR061236 ==
        Adding unknown
        Normalizing (GMPR)
        Computing Bray-Curtis distance
        Performing MDS embedding in 2 dimensions
        KNN machine learning
        Training KNN classifier on 6 cores...
        -> Testing Accuracy: 1.0
        ---------------
        - Sample: SRR061236
                  known:85.01%
                  unknown:14.99%
  == Sample: SRR063471 ==
        Adding unknown
        Normalizing (GMPR)
        Computing Bray-Curtis distance
        Performing MDS embedding in 2 dimensions
        KNN machine learning
        Training KNN classifier on 6 cores...
        -> Testing Accuracy: 1.0
        ---------------
        - Sample: SRR063471
                  known:98.4%
                  unknown:1.6%
  == Sample: SRR1930132 ==
        Adding unknown
        Normalizing (GMPR)
        Computing Bray-Curtis distance
        Performing MDS embedding in 2 dimensions
        KNN machine learning
        Training KNN classifier on 6 cores...
        -> Testing Accuracy: 1.0
        ---------------
        - Sample: SRR1930132
                  known:98.48%
                  unknown:1.52%
  == Sample: SRR1930133 ==
        Adding unknown
        Normalizing (GMPR)
```

(continues on next page)

(continued from previous page)

```
       Computing Bray-Curtis distance
       Performing MDS embedding in 2 dimensions
       KNN machine learning
       Training KNN classifier on 6 cores...
       -> Testing Accuracy: 0.99
       ---------------
       - Sample: SRR1930133
                known:98.49%
                unknown:1.51%
== Sample: SRR7658586 ==
       Adding unknown
       Normalizing (GMPR)
       Computing Bray-Curtis distance
       Performing MDS embedding in 2 dimensions
       KNN machine learning
       Training KNN classifier on 6 cores...
       -> Testing Accuracy: 1.0
       ---------------
       - Sample: SRR7658586
                known:79.65%
                unknown:20.35%
== Sample: SRR7658645 ==
       Adding unknown
       Normalizing (GMPR)
       Computing Bray-Curtis distance
       Performing MDS embedding in 2 dimensions
       KNN machine learning
       Training KNN classifier on 6 cores...
       -> Testing Accuracy: 0.99
       ---------------
       - Sample: SRR7658645
                known:26.88%
                unknown:73.12%
== Sample: SRR7658584 ==
       Adding unknown
       Normalizing (GMPR)
       Computing Bray-Curtis distance
       Performing MDS embedding in 2 dimensions
       KNN machine learning
       Training KNN classifier on 6 cores...
       -> Testing Accuracy: 0.98
       ---------------
       - Sample: SRR7658584
                known:85.78%
                unknown:14.22%
== Sample: SRR7658607 ==
       Adding unknown
       Normalizing (GMPR)
       Computing Bray-Curtis distance
       Performing MDS embedding in 2 dimensions
       KNN machine learning
       Training KNN classifier on 6 cores...
       -> Testing Accuracy: 0.99
       ---------------
       - Sample: SRR7658607
                known:98.74%
                unknown:1.26%
```

(continues on next page)

(continued from previous page)

```
== Sample: SRR7658597 ==
      Adding unknown
      Normalizing (GMPR)
      Computing Bray-Curtis distance
      Performing MDS embedding in 2 dimensions
      KNN machine learning
      Training KNN classifier on 6 cores...
      -> Testing Accuracy: 0.98
      ---------------
      - Sample: SRR7658597
               known:99.1%
               unknown:0.9%
== Sample: SRR5898944 ==
      Adding unknown
      Normalizing (GMPR)
      Computing Bray-Curtis distance
      Performing MDS embedding in 2 dimensions
      KNN machine learning
      Training KNN classifier on 6 cores...
      -> Testing Accuracy: 1.0
      ---------------
      - Sample: SRR5898944
               known:98.48%
               unknown:1.52%
== Sample: ERR1914439 ==
      Adding unknown
      Normalizing (GMPR)
      Computing Bray-Curtis distance
      Performing MDS embedding in 2 dimensions
      KNN machine learning
      Training KNN classifier on 6 cores...
      -> Testing Accuracy: 1.0
      ---------------
      - Sample: ERR1914439
               known:98.48%
               unknown:1.52%
== Sample: ERR1915140 ==
      Adding unknown
      Normalizing (GMPR)
      Computing Bray-Curtis distance
      Performing MDS embedding in 2 dimensions
      KNN machine learning
      Training KNN classifier on 6 cores...
      -> Testing Accuracy: 1.0
      ---------------
      - Sample: ERR1915140
               known:98.48%
               unknown:1.52%
== Sample: ERR1914041 ==
      Adding unknown
      Normalizing (GMPR)
      Computing Bray-Curtis distance
      Performing MDS embedding in 2 dimensions
      KNN machine learning
      Training KNN classifier on 6 cores...
      -> Testing Accuracy: 1.0
      ---------------
```

(continues on next page)

(continued from previous page)

```
       - Sample: ERR1914041
                known:98.48%
                unknown:1.52%
== Sample: ERR1915022 ==
       Adding unknown
       Normalizing (GMPR)
       Computing Bray-Curtis distance
       Performing MDS embedding in 2 dimensions
       KNN machine learning
       Training KNN classifier on 6 cores...
       -> Testing Accuracy: 1.0
       ---------------
       - Sample: ERR1915022
                known:98.48%
                unknown:1.52%
== Sample: ERR1915826 ==
       Adding unknown
       Normalizing (GMPR)
       Computing Bray-Curtis distance
       Performing MDS embedding in 2 dimensions
       KNN machine learning
       Training KNN classifier on 6 cores...
       -> Testing Accuracy: 1.0
       ---------------
       - Sample: ERR1915826
                known:98.48%
                unknown:1.52%
== Sample: ERR1913400 ==
       Adding unknown
       Normalizing (GMPR)
       Computing Bray-Curtis distance
       Performing MDS embedding in 2 dimensions
       KNN machine learning
       Training KNN classifier on 6 cores...
       -> Testing Accuracy: 1.0
       ---------------
       - Sample: ERR1913400
                known:98.48%
                unknown:1.52%
== Sample: ERR1915765 ==
       Adding unknown
       Normalizing (GMPR)
       Computing Bray-Curtis distance
       Performing MDS embedding in 2 dimensions
       KNN machine learning
       Training KNN classifier on 6 cores...
       -> Testing Accuracy: 1.0
       ---------------
       - Sample: ERR1915765
                known:98.48%
                unknown:1.52%
== Sample: ERR1915225 ==
       Adding unknown
       Normalizing (GMPR)
       Computing Bray-Curtis distance
       Performing MDS embedding in 2 dimensions
       KNN machine learning
```

(continues on next page)

(continued from previous page)

```
        Training KNN classifier on 6 cores...
        -> Testing Accuracy: 1.0
        ---------------
        - Sample: ERR1915225
                known:98.48%
                unknown:1.52%
  == Sample: mgm4477874_3 ==
        Adding unknown
        Normalizing (GMPR)
        Computing Bray-Curtis distance
        Performing MDS embedding in 2 dimensions
        KNN machine learning
        Training KNN classifier on 6 cores...
        -> Testing Accuracy: 1.0
        ---------------
        - Sample: mgm4477874_3
                known:72.37%
                unknown:27.63%
  == Sample: ERR1939166 ==
        Adding unknown
        Normalizing (GMPR)
        Computing Bray-Curtis distance
        Performing MDS embedding in 2 dimensions
        KNN machine learning
        Training KNN classifier on 6 cores...
        -> Testing Accuracy: 0.97
        ---------------
        - Sample: ERR1939166
                known:47.44%
                unknown:52.56%
  == Sample: ERR1939165 ==
        Adding unknown
        Normalizing (GMPR)
        Computing Bray-Curtis distance
        Performing MDS embedding in 2 dimensions
        KNN machine learning
        Training KNN classifier on 6 cores...
        -> Testing Accuracy: 0.97
        ---------------
        - Sample: ERR1939165
                known:55.27%
                unknown:44.73%
Step 2: Checking for source proportion
        Computing weighted_unifrac distance on species rank
        TSNE embedding in 2 dimensions
        KNN machine learning
        Performing 5 fold cross validation on 6 cores...
        Trained KNN classifier with 10 neighbors
        -> Testing Accuracy: 0.99
        ---------------
        - Sample: SRR1175007
                Canis_familiaris:1.81%
                Homo_sapiens:96.72%
                Soil:1.47%
        - Sample: SRR061236
                Canis_familiaris:1.81%
                Homo_sapiens:96.72%
```

(continues on next page)

(continued from previous page)

```
        Soil:1.47%
- Sample: SRR063471
        Canis_familiaris:1.81%
        Homo_sapiens:96.72%
        Soil:1.47%
- Sample: SRR1930132
        Canis_familiaris:1.81%
        Homo_sapiens:96.72%
        Soil:1.47%
- Sample: SRR1930133
        Canis_familiaris:1.81%
        Homo_sapiens:96.72%
        Soil:1.47%
- Sample: SRR7658586
        Canis_familiaris:1.81%
        Homo_sapiens:96.72%
        Soil:1.47%
- Sample: SRR7658645
        Canis_familiaris:1.81%
        Homo_sapiens:96.72%
        Soil:1.47%
- Sample: SRR7658584
        Canis_familiaris:1.81%
        Homo_sapiens:96.72%
        Soil:1.47%
- Sample: SRR7658607
        Canis_familiaris:1.81%
        Homo_sapiens:96.72%
        Soil:1.47%
- Sample: SRR7658597
        Canis_familiaris:1.81%
        Homo_sapiens:96.72%
        Soil:1.47%
- Sample: SRR5898944
        Canis_familiaris:1.81%
        Homo_sapiens:96.72%
        Soil:1.47%
- Sample: ERR1914439
        Canis_familiaris:94.25%
        Homo_sapiens:4.28%
        Soil:1.47%
- Sample: ERR1915140
        Canis_familiaris:94.25%
        Homo_sapiens:4.28%
        Soil:1.47%
- Sample: ERR1914041
        Canis_familiaris:94.25%
        Homo_sapiens:4.28%
        Soil:1.47%
- Sample: ERR1915022
        Canis_familiaris:94.25%
        Homo_sapiens:4.28%
        Soil:1.47%
- Sample: ERR1915826
        Canis_familiaris:94.25%
        Homo_sapiens:4.28%
        Soil:1.47%
```

(continues on next page)

(continued from previous page)

```
        - Sample: ERR1913400
                Canis_familiaris:94.25%
                Homo_sapiens:4.28%
                Soil:1.47%
        - Sample: ERR1915765
                Canis_familiaris:94.25%
                Homo_sapiens:4.28%
                Soil:1.47%
        - Sample: ERR1915225
                Canis_familiaris:94.25%
                Homo_sapiens:4.28%
                Soil:1.47%
        - Sample: mgm4477874_3
                Canis_familiaris:2.51%
                Homo_sapiens:5.95%
                Soil:91.53%
        - Sample: ERR1939166
                Canis_familiaris:2.51%
                Homo_sapiens:5.95%
                Soil:91.53%
        - Sample: ERR1939165
                Canis_familiaris:2.51%
                Homo_sapiens:5.95%
                Soil:91.53%
Sourcepredict result written to gut_species_sinks.sourcepredict.csv
Embedding coordinates written to example_embedding.csv
CPU times: user 3.64 s, sys: 828 ms, total: 4.47 s
Wall time: 4min 2s
```

Two files were generated by Sourcepredict: - `gut_species_sinks.sourcepredict.csv` which contains the proportions of each source

[9]:
```
sourcepred = pd.read_csv("gut_species_sinks.sourcepredict.csv", index_col=0)
```

[10]:
```
sourcepred
```

[10]:

	SRR1175007	SRR061236	SRR063471	SRR1930132	SRR1930133	\
Canis_familiaris	0.017814	0.015347	0.017765	0.017779	0.017781	
Homo_sapiens	0.954443	0.822254	0.951788	0.952581	0.952644	
Soil	0.014516	0.012506	0.014476	0.014488	0.014489	
unknown	0.013226	0.149893	0.015971	0.015152	0.015086	

	SRR7658586	SRR7658645	SRR7658584	SRR7658607	SRR7658597	\
Canis_familiaris	0.014379	0.004853	0.015486	0.017825	0.017891	
Homo_sapiens	0.770401	0.260037	0.829699	0.955022	0.958548	
Soil	0.011717	0.003955	0.012619	0.014525	0.014579	
unknown	0.203503	0.731155	0.142196	0.012627	0.008983	

	...	ERR1915140	ERR1914041	ERR1915022	ERR1915826	\
Canis_familiaris	...	0.928216	0.928216	0.928216	0.928216	
Homo_sapiens	...	0.042128	0.042128	0.042128	0.042128	
Soil	...	0.014504	0.014504	0.014504	0.014504	
unknown	...	0.015152	0.015152	0.015152	0.015152	

	ERR1913400	ERR1915765	ERR1915225	mgm4477874_3	\
Canis_familiaris	0.928216	0.928216	0.928216	0.018200	
Homo_sapiens	0.042128	0.042128	0.042128	0.043078	

(continues on next page)

(continued from previous page)

```
Soil                0.014504    0.014504    0.014504      0.662431
unknown             0.015152    0.015152    0.015152      0.276292

                  ERR1939166  ERR1939165
Canis_familiaris    0.011931    0.013898
Homo_sapiens        0.028241    0.032896
Soil                0.434275    0.505859
unknown             0.525554    0.447347

[4 rows x 22 columns]
```

Let's check which organism was predicted for each samples, and compare it with the true source

[11]:
```
comparison = sourcepred.idxmax().to_frame(name="prediction").merge(test_labels, left_
↪index=True, right_index=True)
cm = pdml.ConfusionMatrix(y_true=comparison['labels'],y_pred=comparison['prediction'])
```

```
/Users/borry/miniconda3/lib/python3.6/site-packages/pandas/core/indexing.py:1494:␣
↪FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/indexing.
↪html#deprecate-loc-reindex-listlike
  return self._getitem_tuple(key)
```

Let's look at the confusion matrix

[30]:
```
cm.to_dataframe()
```

[30]:
```
Predicted         Canis_familiaris  Homo_sapiens  Soil  unknown
Actual
Canis_familiaris                 8             0     0        0
Homo_sapiens                     0            10     0        1
Soil                             0             0     2        1
unknown                          0             0     0        0
```

Finally, let's compute the accuracy

[31]:
```
round(cm.stats()['overall']['Accuracy'],2)
```

[31]:
```
0.91
```

91% of the sink samples were correctly predicted !

- The second file generated by sourcepredict is `example_embedding.csv` which contains the embedding coordinates of all samples (sources and sinks)

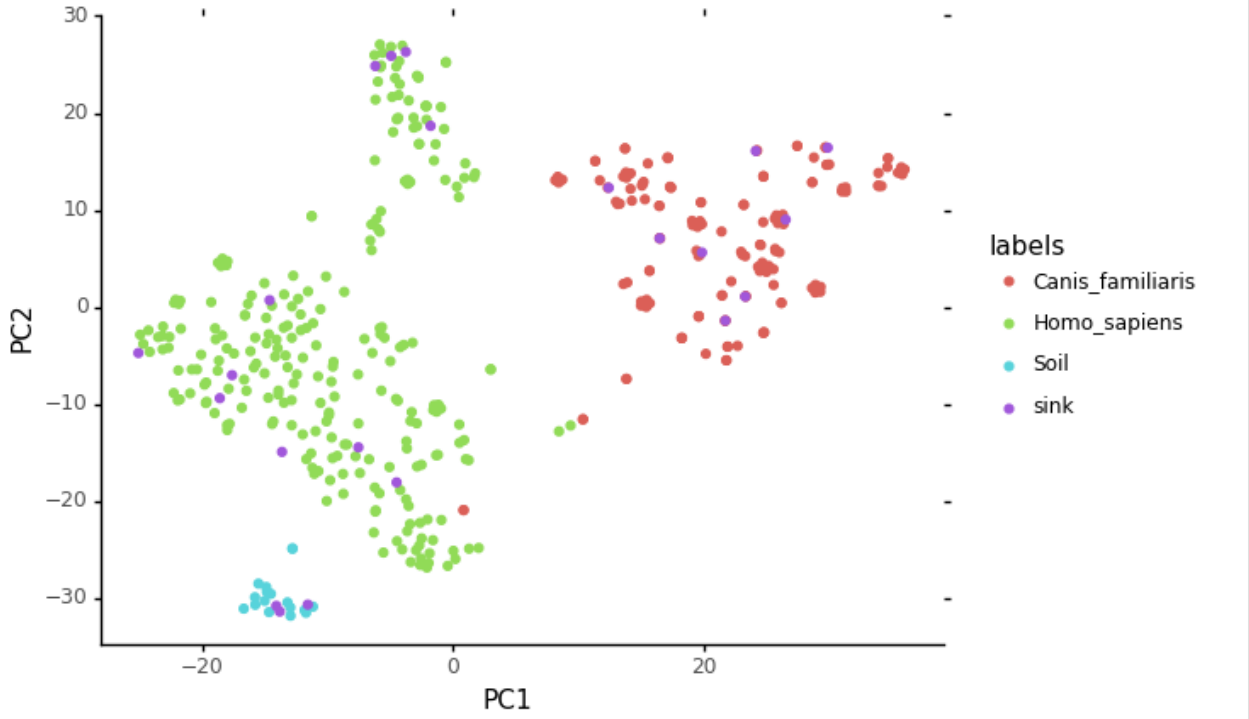
[14]:
```
embed = pd.read_csv("example_embedding.csv", index_col=0)
embed.head()
```

[14]:
```
                  PC1        PC2            labels        name
SRR1761672 -12.706208  -7.860738      Homo_sapiens  SRR1761672
SRR061456   -4.520492  24.795073      Homo_sapiens   SRR061456
SRR1761718 -20.427488  -6.425568      Homo_sapiens  SRR1761718
SRR7658589 -23.176891  -2.985772      Homo_sapiens  SRR7658589
ERR1914932  28.669333  12.863045  Canis_familiaris  ERR1914932
```

We can plot this embedding, using for example, plotnine, which implements the grammar of graphics in Python

[15]:
```
from plotnine import *
import warnings
warnings.filterwarnings('ignore')
```

[16]:
```
ggplot(data = embed, mapping = aes(x="PC1",y="PC2", color="labels")) + geom_point() +␣
↪theme_classic()
```



[16]:
```
<ggplot: (-9223372029842878797)>
```

We can see on this plot where the sink samples were embedded

8.3 Sourcetracker2

"SourceTracker is designed to predict the source of microbial communities in a set of input samples" and is generally consired as the gold standard method to do so. The version 2 is a rewrite of the original Sourcetracker in Python.

We'll reuse the same training and test files, but we need to reformat them a bit for sourcetracker: - In sourcetracker, the source (training) and sink (file) TAXIDs count table is a single file - The metadata file is slightly different

[17]:
```
cnt.to_csv("gut_species_taxid.csv", sep="\t", index_label="TAXID")
```

[18]:
```
test_labels['SourceSink'] = ['sink']*test_labels.shape[0]
```

[19]:
```
train_labels['SourceSink'] = ['source']*train_labels.shape[0]
```

[20]:
```
metadata = train_labels.append(test_labels).rename(columns = {"labels":"Env"})[[
↪'SourceSink','Env']]
metadata.head()
```

[20]:
```
           SourceSink               Env
SRR1761672     source      Homo_sapiens
SRR061456      source      Homo_sapiens
SRR1761718     source      Homo_sapiens
SRR7658589     source      Homo_sapiens
ERR1914932     source  Canis_familiaris
```

[21]:
```
metadata.to_csv("st_gut_species_metadata.csv", sep="\t", index_label='#SampleID')
```

Finally, we need to convert the TAXIDs count table to biom format

[22]:
```
!biom convert -i gut_species_taxid.csv -o gut_species_taxid.biom --table-type="Taxon␣
↪table" --to-json
```

Soucetracker launch command: `sourcetracker2 gibbs -i gut_species_taxid.biom -m st_gut_species_metadata.csv -o gut_species --jobs 6`
(Sourcetracker2 was run on a Linux remote server because of issues running it on MacOS)

[32]:
```
st_pred = pd.read_csv("gut_species/mixing_proportions.txt", sep = "\t", index_col=0)
st_pred.head()
```

[32]:
```
            Canis_familiaris  Homo_sapiens    Soil  Unknown
#SampleID
SRR1175007            0.0170        0.9609  0.0063   0.0158
SRR061236             0.0358        0.9365  0.0074   0.0203
SRR063471             0.0121        0.9724  0.0032   0.0123
SRR1930132            0.1466        0.3761  0.4477   0.0296
SRR1930133            0.1182        0.5082  0.3507   0.0229
```

[34]:
```
st_comparison = st_pred.idxmax(axis=1).to_frame(name="prediction")
st_comparison.head()
```

[34]:
```
              prediction
#SampleID
SRR1175007  Homo_sapiens
SRR061236   Homo_sapiens
SRR063471   Homo_sapiens
SRR1930132          Soil
SRR1930133  Homo_sapiens
```

Let's compare the SourceTracker prediction with the true source

[35]:
```
comparison2 = st_comparison.merge(test_labels, left_index=True, right_index=True)
comparison2
```

[35]:
```
                  prediction            labels SourceSink
SRR1175007        Homo_sapiens      Homo_sapiens       sink
SRR061236         Homo_sapiens      Homo_sapiens       sink
SRR063471         Homo_sapiens      Homo_sapiens       sink
SRR1930132                Soil      Homo_sapiens       sink
SRR1930133        Homo_sapiens      Homo_sapiens       sink
```

(continues on next page)

(continued from previous page)

```
SRR7658586        Homo_sapiens      Homo_sapiens       sink
SRR7658645        Homo_sapiens      Homo_sapiens       sink
SRR7658584                Soil      Homo_sapiens       sink
SRR7658607        Homo_sapiens      Homo_sapiens       sink
SRR7658597        Homo_sapiens      Homo_sapiens       sink
SRR5898944        Homo_sapiens      Homo_sapiens       sink
ERR1914439    Canis_familiaris  Canis_familiaris       sink
ERR1915140                Soil  Canis_familiaris       sink
ERR1914041    Canis_familiaris  Canis_familiaris       sink
ERR1915022    Canis_familiaris  Canis_familiaris       sink
ERR1915826    Canis_familiaris  Canis_familiaris       sink
ERR1913400    Canis_familiaris  Canis_familiaris       sink
ERR1915765    Canis_familiaris  Canis_familiaris       sink
ERR1915225    Canis_familiaris  Canis_familiaris       sink
mgm4477874_3              Soil              Soil       sink
ERR1939166                Soil              Soil       sink
ERR1939165                Soil              Soil       sink
```

Computing the accuracy

[36]:
```
cm2 = pdml.ConfusionMatrix(y_true=comparison2["labels"], y_pred=comparison2[
↪"prediction"])
cm2.to_dataframe()
```

[36]:
```
Predicted         Canis_familiaris  Homo_sapiens  Soil
Actual
Canis_familiaris                 7             0     1
Homo_sapiens                     0             9     2
Soil                             0             0     3
```

[38]:
```
acc2 = round(cm2.stats()['overall']['Accuracy'],2)
```

[38]:
```
0.86
```

Here, Sourcetracker only managed to predict 86% of the sink samples origin correctly

8.4 Conclusion

On this dataset, we've seen that Sourcepredict performs similary or even better than Sourcetracker on predicting accurately the source species

CHAPTER 9

Sourcepredict example2: Estimating source proportions

For this example, we'll reuse the dog, human, and soil dataset.
But unlike example1, here we will mix samples from different sources and estimate the mixing proportions with Sourcepredict and Sourcetracker2

9.1 Preparing mixed samples

[1]:
```
import pandas as pd
from plotnine import *
import numpy as np
```

[2]:
```
cnt = pd.read_csv("../data/modern_gut_microbiomes_sources.csv", index_col=0)
labels = pd.read_csv("../data/modern_gut_microbiomes_labels.csv",index_col=0)
```

As in example 1, we'll first split the dataset into training (95%) and testing(5%)

[3]:
```
cnt_train = cnt.sample(frac=0.95, axis=1)
cnt_test = cnt.drop(cnt_train.columns, axis=1)
train_labels = labels.loc[cnt_train.columns,:]
test_labels = labels.loc[cnt_test.columns,:]
```

[4]:
```
test_labels['labels'].value_counts()
```

[4]:
```
Homo_sapiens        11
Canis_familiaris     9
Soil                 2
Name: labels, dtype: int64
```

[5]:
```
cnt_test.head()
```

[5]:
```
        SRR061456  SRR1175013   SRR059395  SRR1930141  SRR1930247  SRR1761710  \
TAXID
```

(continues on next page)

(continued from previous page)

```
0      14825759.0   4352892.0  13691926.0  27457943.0   1212101.0  24026729.0
6            0.0         0.0       107.0       193.0         0.0        87.0
7            0.0         0.0       107.0       193.0         0.0        87.0
9           96.0       101.0        70.0       412.0         0.0       395.0
10         249.0         0.0       136.0       614.0         0.0       265.0

       SRR1761700  SRR7658605  SRR7658665  SRR7658625  ...  ERR1916299  \
TAXID                                                  ...
0      18876667.0   7776902.0  34166674.0  15983447.0  ...   1481835.0
6            94.0         0.0       215.0       105.0  ...         0.0
7            94.0         0.0       215.0       105.0  ...         0.0
9           199.0       299.0       563.0       369.0  ...        62.0
10          267.0        76.0       985.0       350.0  ...        66.0

       ERR1914213  ERR1915363  ERR1913953  ERR1913947  ERR1916319  ERR1915204  \
TAXID
0       3254064.0   2182037.0   2225689.0   2292745.0   1549960.0    760058.0
6             0.0         0.0         0.0         0.0         0.0         0.0
7             0.0         0.0         0.0         0.0         0.0         0.0
9            63.0       110.0        59.0        63.0        51.0         0.0
10          174.0         0.0        74.0        73.0        62.0         0.0

       ERR1914750  mgm4477803_3  mgm4477877_3
TAXID
0       1750182.0     5492333.0     6004642.0
6             0.0          73.0         216.0
7             0.0          73.0         216.0
9            61.0           0.0           0.0
10           66.0           0.0           0.0

[5 rows x 22 columns]
```

We then create a function to randomly select a sample from each source (dog as s_{dog} and human as s_{human}), and combine such as the new sample $s_{mixed} = p1 * s_{dog} + p1 * s_{human}$

[6]:
```
def create_mixed_sample(cnt, labels, p1, samp_name):
    rand_dog = labels.query('labels == "Canis_familiaris"').sample(1).index[0]
    rand_human = labels.query('labels == "Homo_sapiens"').sample(1).index[0]
    dog_samp = cnt[rand_dog]*p1
    human_samp = cnt[rand_human]*(1-p1)
    comb = dog_samp + human_samp
    comb = comb.rename(samp_name)
    meta = pd.DataFrame({'human_sample':[rand_human],'dog_sample':[rand_dog], 'human_
↪prop':[(1-p1)], 'dog_prop':[p1]}, index=[samp_name])
    return(comb, meta)
```

We run this function for a range of mixed proportions (0 to 90%, by 10%), 3 time for each mix

[7]:
```
mixed_samp = []
mixed_meta = []
nb = 1
for i in range(3):
    for p1 in np.arange(0.1,1,0.1):
        s = create_mixed_sample(cnt=cnt_test, labels=test_labels, p1=p1, samp_name=f
↪"mixed_sample_{nb}")
        mixed_samp.append(s[0])
```

(continues on next page)

(continued from previous page)

```
        mixed_meta.append(s[1])
        nb += 1
```

[8]:
```
mixed_samples = pd.concat(mixed_samp, axis=1, keys=[s.name for s in mixed_samp]).
↪astype(int)
mixed_samples.head()
```

[8]:

TAXID	mixed_sample_1	mixed_sample_2	mixed_sample_3	mixed_sample_4	\
0	1320165	27791888	14189886	14720060	
6	0	172	65	52	
7	0	172	65	52	
9	6	463	158	237	
10	7	802	239	159	

TAXID	mixed_sample_5	mixed_sample_6	mixed_sample_7	mixed_sample_8	\
0	18710369	1414816	2910789	4518936	
6	107	0	0	21	
7	107	0	0	21	
9	313	30	74	61	
10	579	37	51	86	

TAXID	mixed_sample_9	mixed_sample_10	...	mixed_sample_18	mixed_sample_19	\
			...			
0	2282396	7217415	...	5331330	24788154	
6	10	0	...	8	173	
7	10	0	...	8	173	
9	36	280	...	96	370	
10	34	68	...	183	552	

TAXID	mixed_sample_20	mixed_sample_21	mixed_sample_22	mixed_sample_23	\
0	2020394	5968886	3231719	10313424	
6	0	0	0	47	
7	0	0	0	47	
9	132	227	81	130	
10	113	73	24	166	

TAXID	mixed_sample_24	mixed_sample_25	mixed_sample_26	mixed_sample_27
0	8480642	5192549	5175478	4809264
6	37	32	18	19
7	37	32	18	19
9	110	56	88	97
10	144	84	106	127

[5 rows x 27 columns]

[9]:
```
mixed_metadata = pd.concat(mixed_meta)
mixed_metadata.head()
```

[9]:

	human_sample	dog_sample	human_prop	dog_prop
mixed_sample_1	SRR1930247	ERR1913947	0.9	0.1
mixed_sample_2	SRR7658665	ERR1913947	0.8	0.2
mixed_sample_3	SRR1761700	ERR1914213	0.7	0.3
mixed_sample_4	SRR1761710	ERR1915204	0.6	0.4

(continues on next page)

(continued from previous page)

```
mixed_sample_5   SRR7658665  ERR1914213         0.5        0.5
```

Now we can export the new "test" (sink) table to `csv` for sourcepredict

[10]:
```
mixed_samples.to_csv('mixed_samples_cnt.csv')
```

As well as the source count and labels table for the sources

[11]:
```
train_labels.to_csv('train_labels.csv')
cnt_train.to_csv('sources_cnt.csv')
```

9.2 Sourcepredict

For running Sourcepredict, we'll change two parameters from their default values: - `-me` The default method used by Sourcepredict is T-SNE which a non-linear type of embedding, i.e. the distance between points doesn't reflext their actual distance in the original dimensions, to achieve a better clustering, which is good for source prediction. Because here we're more interested in source proportion estimation, rather than source prediction, we'll choose a Multi Dimensional Scaling (MDS) which is a type of linear embedding, where the distance between points in the lozer dimension match more the distances in the embedding in lower dimension, which is better for source proportion estimation. - `-kne` which is the number of neighbors in KNN algorithm: we use a greater (50) number of neighbors to reflect more global contribution of samples to the proportion estimation, instead of only the immediate neighbors. This will affect negatively the source prediction, but give better source proportion estimations - `-kw` which is the *weigth function* in the KNN algorithm. By defaul a **distance** based weight function is apllied to give more weigth to closer samples. However, here, we're more interested in source proportion estimation, rather than source prediction, so we'll disregard the distance based weight function and give the same weight to all neighboring samples, regardless of their distance, with the **uniform** weight function.

[12]:
```
%%time
!python ../sourcepredict -s sources_cnt.csv \
               -l train_labels.csv \
               -n GMPR \
               -kne 50\
               -kw uniform \
               -me MDS \
               -e mixed_embedding.csv \
               -t 6 \
               mixed_samples_cnt.csv
```

```
Step 1: Checking for unknown proportion
  == Sample: mixed_sample_1 ==
        Adding unknown
        Normalizing (GMPR)
        Computing Bray-Curtis distance
        Performing MDS embedding in 2 dimensions
        KNN machine learning
        Training KNN classifier on 6 cores...
        -> Testing Accuracy: 1.0
        ---------------
        - Sample: mixed_sample_1
                  known:98.48%
                  unknown:1.52%
  == Sample: mixed_sample_2 ==
        Adding unknown
```

(continues on next page)

(continued from previous page)

```
      Normalizing (GMPR)
      Computing Bray-Curtis distance
      Performing MDS embedding in 2 dimensions
      KNN machine learning
      Training KNN classifier on 6 cores...
      -> Testing Accuracy: 0.99
      ---------------
      - Sample: mixed_sample_2
               known:99.73%
               unknown:0.27%
== Sample: mixed_sample_3 ==
      Adding unknown
      Normalizing (GMPR)
      Computing Bray-Curtis distance
      Performing MDS embedding in 2 dimensions
      KNN machine learning
      Training KNN classifier on 6 cores...
      -> Testing Accuracy: 1.0
      ---------------
      - Sample: mixed_sample_3
               known:98.79%
               unknown:1.21%
== Sample: mixed_sample_4 ==
      Adding unknown
      Normalizing (GMPR)
      Computing Bray-Curtis distance
      Performing MDS embedding in 2 dimensions
      KNN machine learning
      Training KNN classifier on 6 cores...
      -> Testing Accuracy: 1.0
      ---------------
      - Sample: mixed_sample_4
               known:98.8%
               unknown:1.2%
== Sample: mixed_sample_5 ==
      Adding unknown
      Normalizing (GMPR)
      Computing Bray-Curtis distance
      Performing MDS embedding in 2 dimensions
      KNN machine learning
      Training KNN classifier on 6 cores...
      -> Testing Accuracy: 1.0
      ---------------
      - Sample: mixed_sample_5
               known:98.82%
               unknown:1.18%
== Sample: mixed_sample_6 ==
      Adding unknown
      Normalizing (GMPR)
      Computing Bray-Curtis distance
      Performing MDS embedding in 2 dimensions
      KNN machine learning
      Training KNN classifier on 6 cores...
      -> Testing Accuracy: 1.0
      ---------------
      - Sample: mixed_sample_6
               known:98.48%
```

(continues on next page)

(continued from previous page)

```
                unknown:1.52%
== Sample: mixed_sample_7 ==
       Adding unknown
       Normalizing (GMPR)
       Computing Bray-Curtis distance
       Performing MDS embedding in 2 dimensions
       KNN machine learning
       Training KNN classifier on 6 cores...
       -> Testing Accuracy: 1.0
       ---------------
       - Sample: mixed_sample_7
                known:98.48%
                unknown:1.52%
== Sample: mixed_sample_8 ==
       Adding unknown
       Normalizing (GMPR)
       Computing Bray-Curtis distance
       Performing MDS embedding in 2 dimensions
       KNN machine learning
       Training KNN classifier on 6 cores...
       -> Testing Accuracy: 1.0
       ---------------
       - Sample: mixed_sample_8
                known:98.48%
                unknown:1.52%
== Sample: mixed_sample_9 ==
       Adding unknown
       Normalizing (GMPR)
       Computing Bray-Curtis distance
       Performing MDS embedding in 2 dimensions
       KNN machine learning
       Training KNN classifier on 6 cores...
       -> Testing Accuracy: 1.0
       ---------------
       - Sample: mixed_sample_9
                known:98.48%
                unknown:1.52%
== Sample: mixed_sample_10 ==
       Adding unknown
       Normalizing (GMPR)
       Computing Bray-Curtis distance
       Performing MDS embedding in 2 dimensions
       KNN machine learning
       Training KNN classifier on 6 cores...
       -> Testing Accuracy: 1.0
       ---------------
       - Sample: mixed_sample_10
                known:98.48%
                unknown:1.52%
== Sample: mixed_sample_11 ==
       Adding unknown
       Normalizing (GMPR)
       Computing Bray-Curtis distance
       Performing MDS embedding in 2 dimensions
       KNN machine learning
       Training KNN classifier on 6 cores...
       -> Testing Accuracy: 0.99
```

(continues on next page)

(continued from previous page)

```
      ---------------
      - Sample: mixed_sample_11
                known:99.16%
                unknown:0.84%
== Sample: mixed_sample_12 ==
      Adding unknown
      Normalizing (GMPR)
      Computing Bray-Curtis distance
      Performing MDS embedding in 2 dimensions
      KNN machine learning
      Training KNN classifier on 6 cores...
      -> Testing Accuracy: 1.0
      ---------------
      - Sample: mixed_sample_12
                known:98.5%
                unknown:1.5%
== Sample: mixed_sample_13 ==
      Adding unknown
      Normalizing (GMPR)
      Computing Bray-Curtis distance
      Performing MDS embedding in 2 dimensions
      KNN machine learning
      Training KNN classifier on 6 cores...
      -> Testing Accuracy: 1.0
      ---------------
      - Sample: mixed_sample_13
                known:98.48%
                unknown:1.52%
== Sample: mixed_sample_14 ==
      Adding unknown
      Normalizing (GMPR)
      Computing Bray-Curtis distance
      Performing MDS embedding in 2 dimensions
      KNN machine learning
      Training KNN classifier on 6 cores...
      -> Testing Accuracy: 1.0
      ---------------
      - Sample: mixed_sample_14
                known:98.48%
                unknown:1.52%
== Sample: mixed_sample_15 ==
      Adding unknown
      Normalizing (GMPR)
      Computing Bray-Curtis distance
      Performing MDS embedding in 2 dimensions
      KNN machine learning
      Training KNN classifier on 6 cores...
      -> Testing Accuracy: 1.0
      ---------------
      - Sample: mixed_sample_15
                known:98.49%
                unknown:1.51%
== Sample: mixed_sample_16 ==
      Adding unknown
      Normalizing (GMPR)
      Computing Bray-Curtis distance
      Performing MDS embedding in 2 dimensions
```

(continues on next page)

(continued from previous page)

```
       KNN machine learning
       Training KNN classifier on 6 cores...
       -> Testing Accuracy: 1.0
       ---------------
       - Sample: mixed_sample_16
                known:98.48%
                unknown:1.52%
== Sample: mixed_sample_17 ==
       Adding unknown
       Normalizing (GMPR)
       Computing Bray-Curtis distance
       Performing MDS embedding in 2 dimensions
       KNN machine learning
       Training KNN classifier on 6 cores...
       -> Testing Accuracy: 1.0
       ---------------
       - Sample: mixed_sample_17
                known:98.48%
                unknown:1.52%
== Sample: mixed_sample_18 ==
       Adding unknown
       Normalizing (GMPR)
       Computing Bray-Curtis distance
       Performing MDS embedding in 2 dimensions
       KNN machine learning
       Training KNN classifier on 6 cores...
       -> Testing Accuracy: 1.0
       ---------------
       - Sample: mixed_sample_18
                known:98.48%
                unknown:1.52%
== Sample: mixed_sample_19 ==
       Adding unknown
       Normalizing (GMPR)
       Computing Bray-Curtis distance
       Performing MDS embedding in 2 dimensions
       KNN machine learning
       Training KNN classifier on 6 cores...
       -> Testing Accuracy: 0.9
       ---------------
       - Sample: mixed_sample_19
                known:99.79%
                unknown:0.21%
== Sample: mixed_sample_20 ==
       Adding unknown
       Normalizing (GMPR)
       Computing Bray-Curtis distance
       Performing MDS embedding in 2 dimensions
       KNN machine learning
       Training KNN classifier on 6 cores...
       -> Testing Accuracy: 1.0
       ---------------
       - Sample: mixed_sample_20
                known:98.48%
                unknown:1.52%
== Sample: mixed_sample_21 ==
       Adding unknown
```

(continues on next page)

(continued from previous page)

```
      Normalizing (GMPR)
      Computing Bray-Curtis distance
      Performing MDS embedding in 2 dimensions
      KNN machine learning
      Training KNN classifier on 6 cores...
      -> Testing Accuracy: 1.0
      ---------------
      - Sample: mixed_sample_21
               known:98.48%
               unknown:1.52%
== Sample: mixed_sample_22 ==
      Adding unknown
      Normalizing (GMPR)
      Computing Bray-Curtis distance
      Performing MDS embedding in 2 dimensions
      KNN machine learning
      Training KNN classifier on 6 cores...
      -> Testing Accuracy: 1.0
      ---------------
      - Sample: mixed_sample_22
               known:98.48%
               unknown:1.52%
== Sample: mixed_sample_23 ==
      Adding unknown
      Normalizing (GMPR)
      Computing Bray-Curtis distance
      Performing MDS embedding in 2 dimensions
      KNN machine learning
      Training KNN classifier on 6 cores...
      -> Testing Accuracy: 1.0
      ---------------
      - Sample: mixed_sample_23
               known:98.48%
               unknown:1.52%
== Sample: mixed_sample_24 ==
      Adding unknown
      Normalizing (GMPR)
      Computing Bray-Curtis distance
      Performing MDS embedding in 2 dimensions
      KNN machine learning
      Training KNN classifier on 6 cores...
      -> Testing Accuracy: 1.0
      ---------------
      - Sample: mixed_sample_24
               known:98.48%
               unknown:1.52%
== Sample: mixed_sample_25 ==
      Adding unknown
      Normalizing (GMPR)
      Computing Bray-Curtis distance
      Performing MDS embedding in 2 dimensions
      KNN machine learning
      Training KNN classifier on 6 cores...
      -> Testing Accuracy: 1.0
      ---------------
      - Sample: mixed_sample_25
               known:98.48%
```

(continues on next page)

(continued from previous page)

```
                unknown:1.52%
  == Sample: mixed_sample_26 ==
        Adding unknown
        Normalizing (GMPR)
        Computing Bray-Curtis distance
        Performing MDS embedding in 2 dimensions
        KNN machine learning
        Training KNN classifier on 6 cores...
        -> Testing Accuracy: 1.0
        ---------------
        - Sample: mixed_sample_26
                known:98.48%
                unknown:1.52%
  == Sample: mixed_sample_27 ==
        Adding unknown
        Normalizing (GMPR)
        Computing Bray-Curtis distance
        Performing MDS embedding in 2 dimensions
        KNN machine learning
        Training KNN classifier on 6 cores...
        -> Testing Accuracy: 1.0
        ---------------
        - Sample: mixed_sample_27
                known:98.48%
                unknown:1.52%
Step 2: Checking for source proportion
        Computing weighted_unifrac distance on species rank
        MDS embedding in 2 dimensions
        KNN machine learning
        Trained KNN classifier with 50 neighbors
        -> Testing Accuracy: 0.88
        ---------------
        - Sample: mixed_sample_1
                Canis_familiaris:87.65%
                Homo_sapiens:11.14%
                Soil:1.21%
        - Sample: mixed_sample_2
                Canis_familiaris:3.66%
                Homo_sapiens:95.03%
                Soil:1.31%
        - Sample: mixed_sample_3
                Canis_familiaris:3.66%
                Homo_sapiens:95.03%
                Soil:1.31%
        - Sample: mixed_sample_4
                Canis_familiaris:5.65%
                Homo_sapiens:93.02%
                Soil:1.33%
        - Sample: mixed_sample_5
                Canis_familiaris:3.66%
                Homo_sapiens:95.03%
                Soil:1.31%
        - Sample: mixed_sample_6
                Canis_familiaris:92.78%
                Homo_sapiens:6.02%
                Soil:1.2%
        - Sample: mixed_sample_7
```

(continues on next page)

(continued from previous page)

```
          Canis_familiaris:54.39%
          Homo_sapiens:44.26%
          Soil:1.35%
      - Sample: mixed_sample_8
          Canis_familiaris:36.94%
          Homo_sapiens:61.66%
          Soil:1.4%
      - Sample: mixed_sample_9
          Canis_familiaris:5.65%
          Homo_sapiens:93.02%
          Soil:1.33%
      - Sample: mixed_sample_10
          Canis_familiaris:5.65%
          Homo_sapiens:93.02%
          Soil:1.33%
      - Sample: mixed_sample_11
          Canis_familiaris:5.65%
          Homo_sapiens:93.02%
          Soil:1.33%
      - Sample: mixed_sample_12
          Canis_familiaris:30.32%
          Homo_sapiens:68.27%
          Soil:1.41%
      - Sample: mixed_sample_13
          Canis_familiaris:5.65%
          Homo_sapiens:93.02%
          Soil:1.33%
      - Sample: mixed_sample_14
          Canis_familiaris:30.32%
          Homo_sapiens:68.27%
          Soil:1.41%
      - Sample: mixed_sample_15
          Canis_familiaris:24.34%
          Homo_sapiens:74.24%
          Soil:1.41%
      - Sample: mixed_sample_16
          Canis_familiaris:30.32%
          Homo_sapiens:68.27%
          Soil:1.41%
      - Sample: mixed_sample_17
          Canis_familiaris:27.24%
          Homo_sapiens:71.35%
          Soil:1.41%
      - Sample: mixed_sample_18
          Canis_familiaris:83.76%
          Homo_sapiens:15.02%
          Soil:1.22%
      - Sample: mixed_sample_19
          Canis_familiaris:24.34%
          Homo_sapiens:74.24%
          Soil:1.41%
      - Sample: mixed_sample_20
          Canis_familiaris:3.66%
          Homo_sapiens:95.03%
          Soil:1.31%
      - Sample: mixed_sample_21
          Canis_familiaris:5.65%
```

(continues on next page)

(continued from previous page)

```
                Homo_sapiens:93.02%
                Soil:1.33%
        - Sample: mixed_sample_22
                Canis_familiaris:40.4%
                Homo_sapiens:58.2%
                Soil:1.4%
        - Sample: mixed_sample_23
                Canis_familiaris:3.66%
                Homo_sapiens:95.03%
                Soil:1.31%
        - Sample: mixed_sample_24
                Canis_familiaris:3.66%
                Homo_sapiens:95.03%
                Soil:1.31%
        - Sample: mixed_sample_25
                Canis_familiaris:30.32%
                Homo_sapiens:68.27%
                Soil:1.41%
        - Sample: mixed_sample_26
                Canis_familiaris:21.65%
                Homo_sapiens:76.94%
                Soil:1.41%
        - Sample: mixed_sample_27
                Canis_familiaris:85.18%
                Homo_sapiens:13.6%
                Soil:1.22%
Sourcepredict result written to mixed_samples_cnt.sourcepredict.csv
Embedding coordinates written to mixed_embedding.csv
CPU times: user 4.84 s, sys: 1.13 s, total: 5.97 s
Wall time: 5min 14s
```

Reading Sourcepredict results

[13]:
```
sp_ebd = pd.read_csv("mixed_embedding.csv", index_col=0)
```

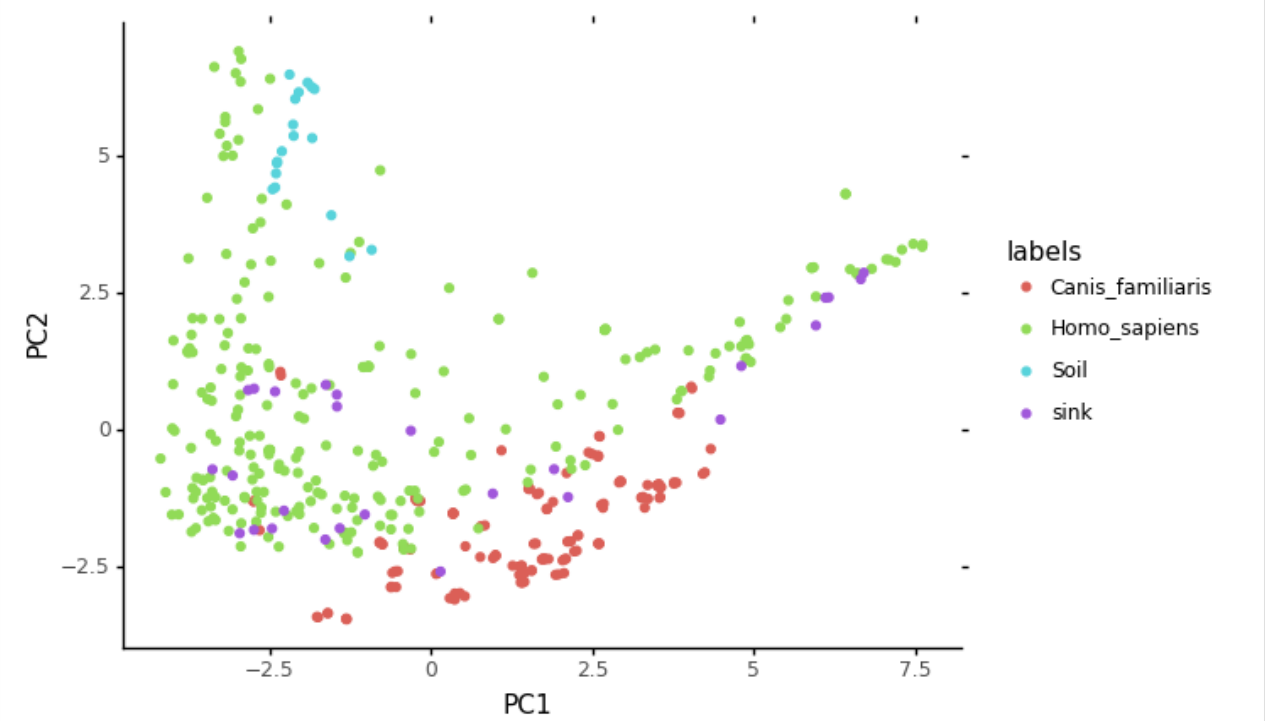
[14]:
```
sp_ebd.head()
```

[14]:

	PC1	PC2	labels	name
SRR1761712	-3.713176	-0.344326	Homo_sapiens	SRR1761712
ERR1913614	2.586560	-0.498098	Canis_familiaris	ERR1913614
ERR1914349	-1.595982	-3.363762	Canis_familiaris	ERR1914349
SRR1930255	2.174966	-0.728862	Homo_sapiens	SRR1930255
SRR1646027	-3.329213	-0.214682	Homo_sapiens	SRR1646027

[15]:
```
import warnings
warnings.filterwarnings('ignore')
```

[16]:
```
ggplot(data = sp_ebd, mapping = aes(x='PC1',y='PC2')) + geom_point(aes(color='labels
↪')) + theme_classic()
```



[16]:
```
<ggplot: (-9223372029299469603)>
```

[17]:
```
sp_pred = pd.read_csv("mixed_samples_cnt.sourcepredict.csv", index_col=0)
```

[18]:
```
sp_pred.T.head()
```

[18]:

	Canis_familiaris	Homo_sapiens	Soil	unknown
mixed_sample_1	0.863266	0.109678	0.011905	0.015152
mixed_sample_2	0.036542	0.947680	0.013046	0.002733
mixed_sample_3	0.036198	0.938776	0.012923	0.012103
mixed_sample_4	0.055810	0.919077	0.013163	0.011951
mixed_sample_5	0.036211	0.939098	0.012927	0.011764

[19]:
```
mixed_metadata.head()
```

[19]:

	human_sample	dog_sample	human_prop	dog_prop
mixed_sample_1	SRR1930247	ERR1913947	0.9	0.1
mixed_sample_2	SRR7658665	ERR1913947	0.8	0.2
mixed_sample_3	SRR1761700	ERR1914213	0.7	0.3
mixed_sample_4	SRR1761710	ERR1915204	0.6	0.4
mixed_sample_5	SRR7658665	ERR1914213	0.5	0.5

[20]:
```
sp_res = sp_pred.T.merge(mixed_metadata, left_index=True, right_index=True)
```

[21]:
```
sp_res.head()
```

[21]:

	Canis_familiaris	Homo_sapiens	Soil	unknown	\
mixed_sample_1	0.863266	0.109678	0.011905	0.015152	
mixed_sample_2	0.036542	0.947680	0.013046	0.002733	
mixed_sample_3	0.036198	0.938776	0.012923	0.012103	
mixed_sample_4	0.055810	0.919077	0.013163	0.011951	

(continues on next page)

(continued from previous page)

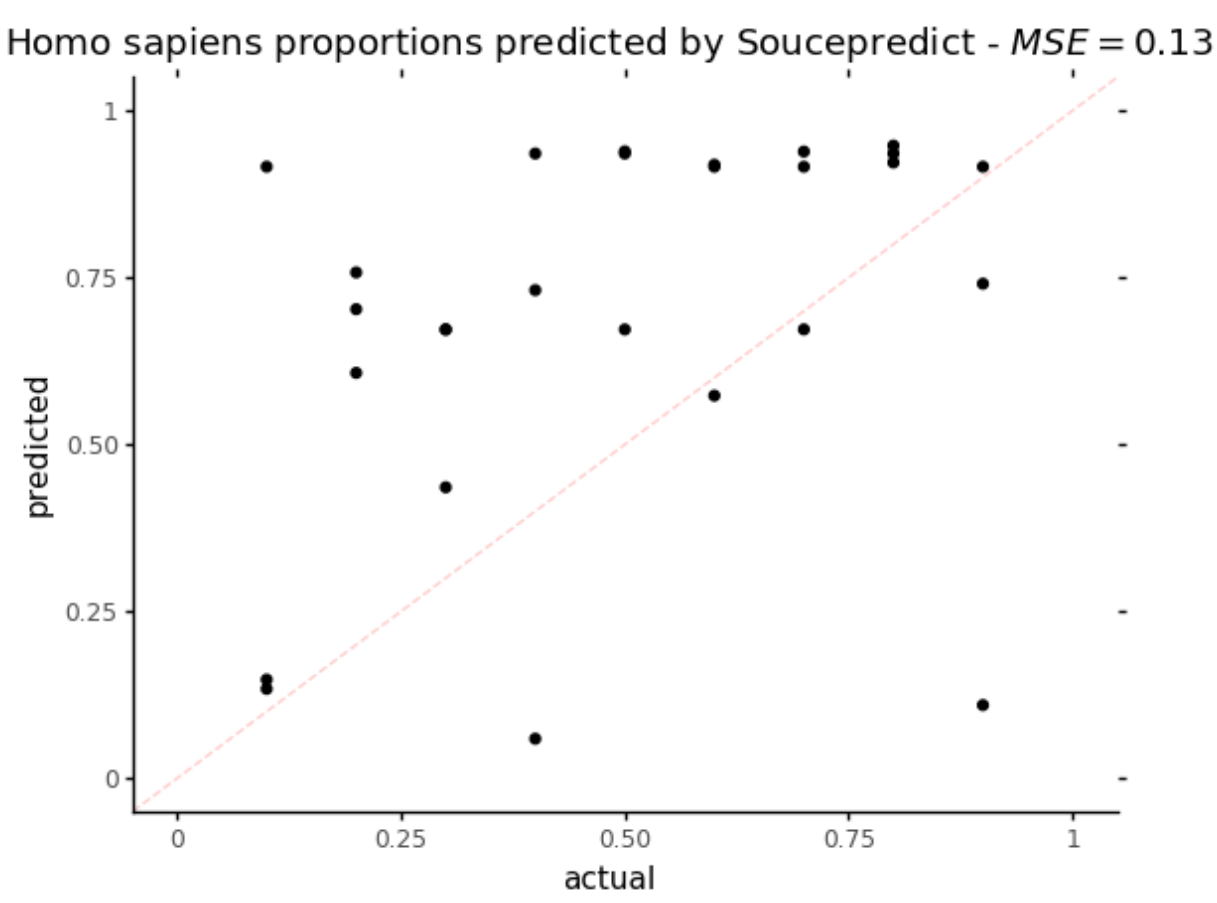
```
mixed_sample_5          0.036211      0.939098  0.012927  0.011764

               human_sample  dog_sample  human_prop  dog_prop
mixed_sample_1   SRR1930247  ERR1913947         0.9       0.1
mixed_sample_2   SRR7658665  ERR1913947         0.8       0.2
mixed_sample_3   SRR1761700  ERR1914213         0.7       0.3
mixed_sample_4   SRR1761710  ERR1915204         0.6       0.4
mixed_sample_5   SRR7658665  ERR1914213         0.5       0.5
```

[22]:
```
from sklearn.metrics import r2_score, mean_squared_error
```

[23]:
```
mse_sp = round(mean_squared_error(y_pred=sp_res['Homo_sapiens'], y_true=sp_res['human_
↪prop']),2)
```

[24]:
```
p = ggplot(data = sp_res, mapping=aes(x='human_prop',y='Homo_sapiens')) + geom_point()
p += labs(title = f"Homo sapiens proportions predicted by Soucepredict - $MSE = {mse_
↪sp}$", x='actual', y='predicted')
p += theme_classic()
p += coord_cartesian(xlim=[0,1], ylim=[0,1])
p += geom_abline(intercept=0, slope=1, color = "red", alpha=0.2, linetype = 'dashed')
p
```



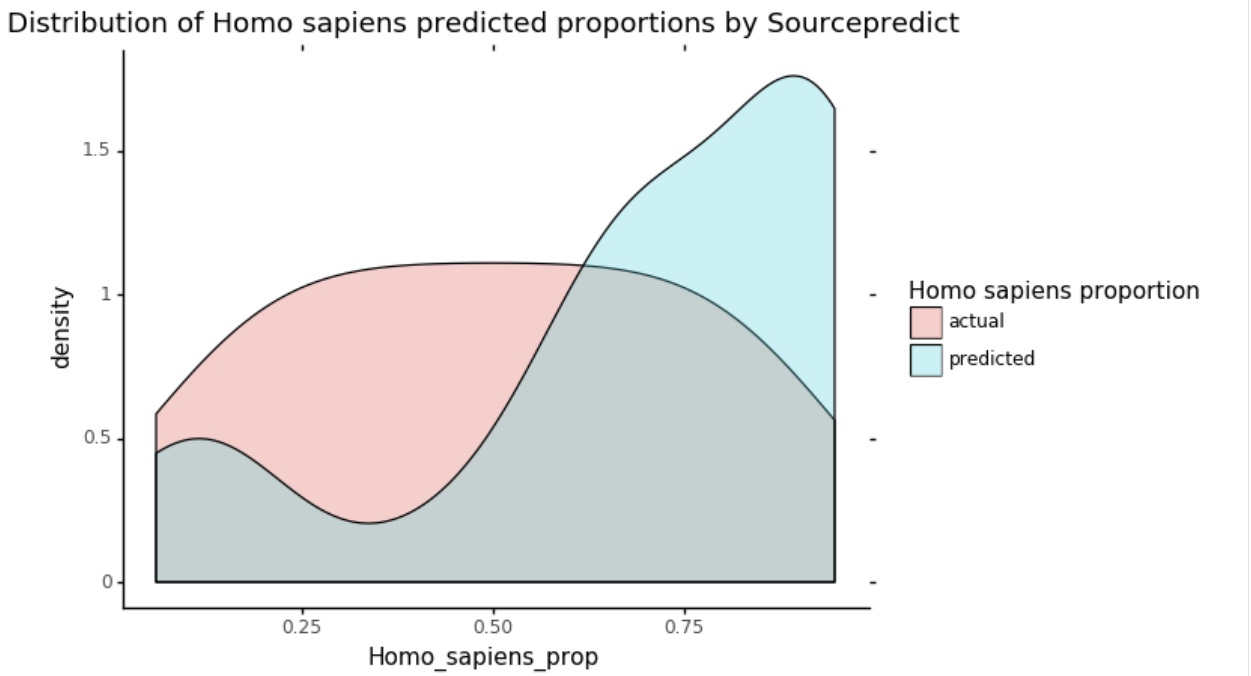
[24]:
```
<ggplot: (-9223372036555534967)>
```

On this plot, the dotted red line represents what a perfect proportion estimation would give, with a Mean Squared Error

(MSE) = 0.

[25]:
```
sp_res_hist = (sp_res['human_prop'].append(sp_res['Homo_sapiens']).to_frame(name=
↪'Homo_sapiens_prop'))
sp_res_hist['source'] = (['actual']*sp_res.shape[0]+['predicted']*sp_res.shape[0])
```

[47]:
```
p = ggplot(data = sp_res_hist, mapping=aes(x='Homo_sapiens_prop')) + geom_
↪density(aes(fill='source'), alpha=0.3)
p += labs(title = 'Distribution of Homo sapiens predicted proportions by Sourcepredict
↪')
p += scale_fill_discrete(name="Homo sapiens proportion")
p += theme_classic()
p
```



[47]:
```
<ggplot: (-9223372029298930965)>
```

This plot shows the actual and predicted by Sourcepredict distribution of Human proportions. What we are interested in is the overlap between the two colors: the higer it is, the more the estimated Human proportion is accurate.

9.3 Sourcetracker2

Preparing count table

[27]:
```
cnt_train.merge(mixed_samples, right_index=True, left_index=True).to_csv("st_mixed_
↪count.csv" , sep="\t", index_label="TAXID")
```

[28]:
```
!biom convert -i st_mixed_count.csv -o st_mixed_count.biom --table-type="Taxon table"␣
↪--to-json
```

Preparing metadata

[29]:
```
train_labels['SourceSink'] = ['source']*train_labels.shape[0]
```

[30]:
```
mixed_metadata['labels'] = ['-']*mixed_metadata.shape[0]
mixed_metadata['SourceSink'] = ['sink']*mixed_metadata.shape[0]
```

[31]:
```
st_labels = train_labels.append(mixed_metadata[['labels', 'SourceSink']])
```

[32]:
```
st_labels = st_labels.rename(columns={'labels':'Env'})[['SourceSink','Env']]
```

[33]:
```
st_labels.to_csv("st_mixed_labels.csv", sep="\t", index_label='#SampleID')
```

Running Sourcetracker2 `sourcetracker2 gibbs -i st_mixed_count.biom -m st_mixed_labels.csv -o mixed_prop --jobs 6`
(Sourcetracker2 was run on a Linux remote server because of issues running it on MacOS)

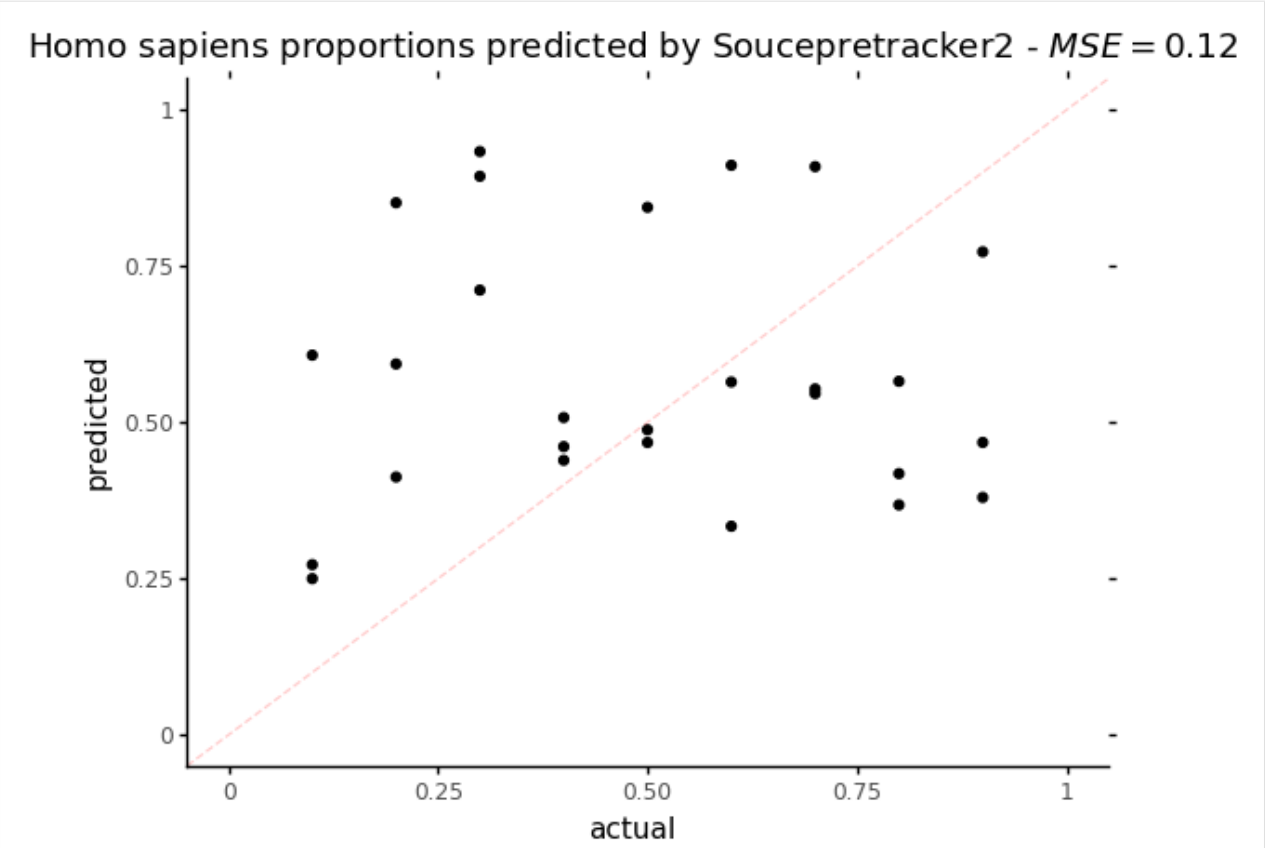
Sourcetracker2 results

[40]:
```
st_pred = pd.read_csv("mixed_prop/mixing_proportions.txt", sep="\t", index_col=0)
```

[41]:
```
st_res = st_pred.merge(mixed_metadata, left_index=True, right_index=True)
```

[42]:
```
mse_st = round(mean_squared_error(y_pred=st_res['Homo_sapiens'], y_true=st_res['human_
↪prop']),2)
```

[43]:
```
p = ggplot(data = st_res, mapping=aes(x='human_prop',y='Homo_sapiens')) + geom_point()
p += labs(title = f"Homo sapiens proportions predicted by Soucepretracker2 - $MSE =
↪{mse_st}$", x='actual', y='predicted')
p += theme_classic()
p += coord_cartesian(xlim=[0,1], ylim=[0,1])
p += geom_abline(intercept=0, slope=1, color = "red", alpha=0.2, linetype = 'dashed')
p
```

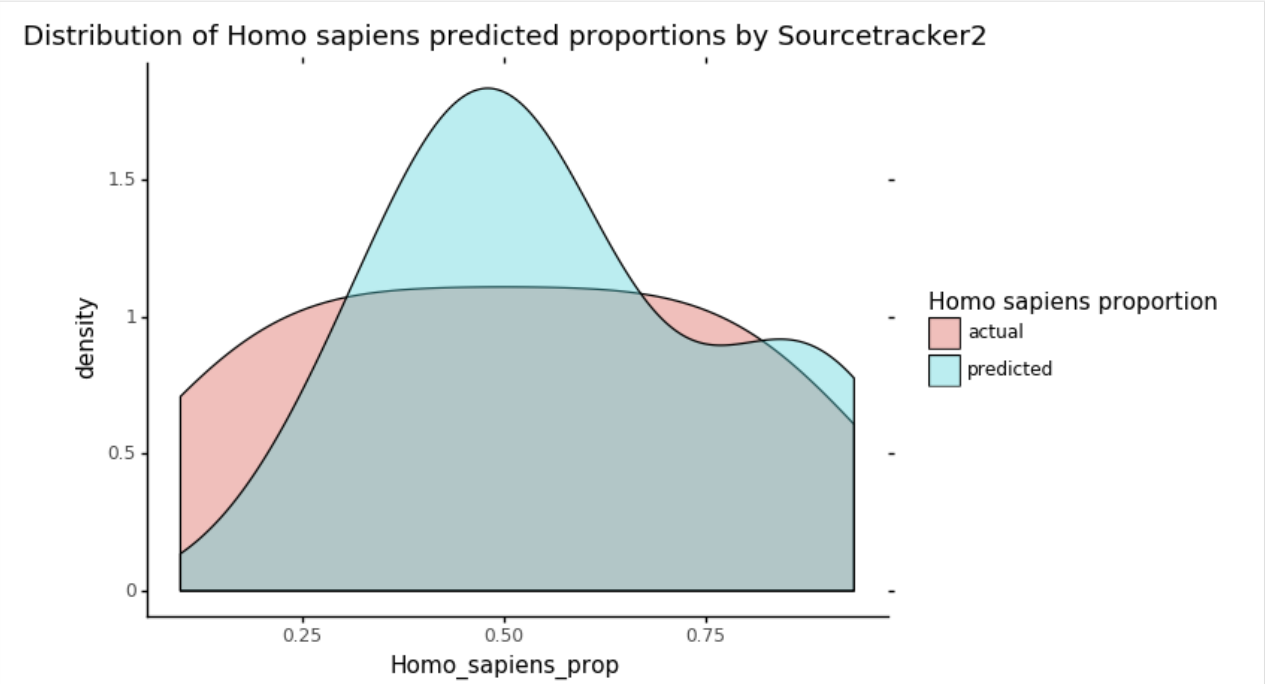


[43]:
```
<ggplot: (-9223372029300899661)>
```

On this plot, the dotted red line represents what a perfect proportion estimation would give, with a Mean Squared Error (MSE) = 0. Regarding the MSE, Sourcepredict and Sourcetracker perform similarly with a MSE of 0.13 for Sourcepredict and 0.12 for Sourcetracker.

[44]:
```
st_res_hist = (st_res['human_prop'].append(st_res['Homo_sapiens']).to_frame(name=
↪'Homo_sapiens_prop'))
st_res_hist['source'] = (['actual']*st_res.shape[0]+['predicted']*st_res.shape[0])
```

[46]:
```
p = ggplot(data = st_res_hist, mapping=aes(x='Homo_sapiens_prop')) + geom_
↪density(aes(fill='source'), alpha=0.4)
p += labs(title = 'Distribution of Homo sapiens predicted proportions by␣
↪Sourcetracker2')
p += scale_fill_discrete(name="Homo sapiens proportion")
p += theme_classic()
p
```



[46]: <ggplot: (7555587890)>

This plot shows the actual and predicted by Sourcepredict distribution of Human proportions. What we are interested in is the overlap between the two colors: the higer it is, the more the estimated Human proportion is accurate.

Here, there is a bigger overlap between actual and predicted, suggesting a slightly better source proportion estimation than with Sourcepredict.

9.4 Conclusion

For source proportion estimation in samples of mixed sources, we've seen that Sourcepredict, with adapted parameters, can perform similarly as Sourcetracker.

However, because Sourcepredict was designed for source prediction in mind, as opposed to source proportion estimation, it requires parameters tweaking to achive the same results as Sourcetracker.

Therefore, for source proportion estimation, we still recommend using Sourcetracker, even if Sourcepredict can perform similarly.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search