phenotrex Documentation

Release 0.6.0

Lukas Lüftinger

Sep 10, 2021

CONTENTS:

1 phenotrex **1**
- 1.1 Installation . . . 1
- 1.2 Usage . . . 1

2 Installation **3**
- 2.1 Stable release . . . 3
- 2.2 From sources . . . 3

3 Usage Tutorial **5**
- 3.1 Creation of Phenotrex Input Features . . . 5
- 3.2 Training of Phenotrex Classifiers . . . 6
- 3.3 Performance Estimation of Phenotrex Classifiers . . . 7
- 3.4 Performance Estimation for Metagenomic Phenotrex Classifiers . . . 7
- 3.5 Predicting Phenotypes with Phenotrex . . . 8
- 3.6 Explanation of Phenotrex Predictions . . . 9

4 phenotrex **11**
- 4.1 phenotrex package . . . 11

5 Credits **17**
- 5.1 Development Lead . . . 17
- 5.2 Contributors . . . 17

6 History **19**

7 Indices and tables **21**

Python Module Index **23**

Index **25**

CHAPTER

ONE

PHENOTREX

End-to-end Microbial Phenotypic Trait Prediction.

1.1 Installation

```
$ pip install phenotrex[fasta]
```

1.2 Usage

Phenotrex is a component of the PhenDB web server, which performs phenotypic trait prediction on user-uploaded metagenomic bins. To try out phenotrex with PhenDB's pre-trained and curated set of trait models, genomes may thus simply be submitted to PhenDB.

1.2.1 Basic Usage

To use a trained phenotrex model `MY_TRAIT.pkl` for prediction of a phenotypic trait with a given genome `genome.fna`:

```
$ phenotrex predict --classifier MY_TRAIT.pkl genome.fna > predictions.tsv
```

This yields a tabular file containing a prediction regarding the presence of the trait (YES or NO), as well as a confidence value the model ascribes to this prediction, ranging from 0.5 to 1.0.

1.2.2 Advanced Usage

For training, evaluation and explanation of phenotrex models on user data, please refer to the full usage tutorial here.

CHAPTER

TWO

INSTALLATION

2.1 Stable release

For a full installation of phenotrex, run this command in your terminal:

```
$ pip install phenotrex[fasta]
```

Note that this command installs large dependencies (pytorch, deepnog) required for transforming FASTA files into phenotrex input features at runtime.

If this capability is not needed (for example because feature files have been pre-created), installation size can be significantly reduced by running instead:

```
$ pip install phenotrex
```

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for phenotrex can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/univieCUBE/phenotrex
```

Or download the tarball:

```
$ curl -OL https://github.com/univieCUBE/phenotrex/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ make full-install
```

CHAPTER

THREE

USAGE TUTORIAL

The following tutorial illustrates training, evaluation, inference and model introspection with phenotrex using the phenotrex command line interface. For further information on flags and parameters used in this tutorial, please consult the relevant CLI documentation available via `phenotrex <command> --help`.

To illustrate phenotrex's capabilities, a small dataset of genome assemblies will be used. To download all required data, run:

```
$ curl -OL http://fileshare.csb.univie.ac.at/phenotrex/tutorial_data.tar
$ tar -xf tutorial_data.tar
$ cd tutorial_data
```

To then install phenotrex, including its capability for extracting features from FASTA files, in a new virtual environment:

```
$ python3 -m venv phenotrex-env
$ source ./phenotrex-env/bin/activate
$ pip install phenotrex[fasta]
```

3.1 Creation of Phenotrex Input Features

Phenotrex operates on presence/absence patterns of eggNOG cluster IDs in the passed genome. If a DNA FASTA file is passed to phenotrex, Prodigal is first used to find protein sequences - this step is skipped if a protein FASTA file is passed instead. To then find eggNOG cluster IDs from protein sequences, deepnog is used. Input files to feature creation may thus be DNA or protein multi-FASTA files, which may optionally be gzipped.

Feature creation is computationally demanding. For this reason, direct input of individual FASTA files to phenotrex is only implemented for prediction. For model training, evaluation (and for batch prediction), tabular files must be created from input FASTA files representing the genotype of all input files, encoded as eggNOG cluster IDs. This allows reuse of the created features for all training and evaluation purposes.

To create a tabular genotype file suitable for use in phenotrex training:

```
$ phenotrex compute-genotype \
    --out T3SS.train_eval.genotype \
    --threads 4 \
    train_eval/genomes/*.fna.gz
```

After some time, this will create a new tab-separated values (TSV) file `T3SS.train_eval.genotype` in the current directory, of the following shape:

```
#feature_type:eggNOG5-tax-2
GCA_003096415.1.fna.gz      COG0656 COG0661 COG3161 COG0358 ...
GCF_000006765.1.fna.gz      COG0593 COG3255 COG1195 COG0187 ...
GCF_000006905.1.fna.gz      COG2202 COG0169 COG0237 COG0847 ...
...
```

Here, all lines starting with # denote a metadata field read by phenotrex - for example the type of features, in this case eggNOG clusters for the NCBI taxon ID 2 (Bacteria) from eggNOG version 5. In each following line, the first field denotes the identifier of the input genome (the file name), followed by all features found in the genome, each separated by tabs.

Note: Feature creation by phenotrex scales reasonably well with the number of threads supplied to the `compute-genotype` command (`--threads`). However, for large sets of genomes and when compute cluster resources are available to the user, it may be more expedient to compute subsets of genomes in parallel on different machines, and concatenate them afterwards.

3.2 Training of Phenotrex Classifiers

To train a phenotrex classifier, two tabular input files are required: The `genotype` file (created from FASTA files in the last section), containing representations of the input genomes; and the `phenotype` file, containing true phenotypic trait values for each input genome on which to train and evaluate the model. For this tutorial, we provide a phenotype file containing information on Type 3 secretion system (T3SS) presence in each of the input genomes.

The tabular phenotype file required for training and model evaluation has the following shape:

```
Identifier  T3SS
GCF_000012905.2.fna.gz      NO
GCF_000195735.1.fna.gz      NO
GCF_000060345.1.fna.gz      NO
GCF_000959505.1.fna.gz      YES
GCF_000220235.1.fna.gz      NO
GCF_000190695.1.fna.gz      NO
GCF_000007605.1.fna.gz      YES
GCF_000195995.1.fna.gz      YES
GCF_000015365.1.fna.gz      NO
GCF_000173115.1.fna.gz      NO
GCF_000173095.1.fna.gz      NO
GCA_003096415.1.fna.gz      NO
...
```

The first column of the file contains identifiers (file names) mapping to those in the genotype file, and the second column contains true phenotypic trait values. During training, the model will store the header of column 2 as the name of the trait.

Phenotrex implements model training using two different machine learning algorithms: XGBoost (XGB) and Support Vector Machine (SVM). For each algorithm, a number of hyperparameters are settable for training and evaluation. Please consult the output of `phenotrex train xgb --help` and `phenotrex train svm --help`, as well as the relevant documentation of the underlying implementations. When no hyperparameters are selected, reasonable (but possibly suboptimal) defaults are used.

To train an XGB classifier with the previously created genotype and the given phenotype file:

```
$ phenotrex train xgb \
    --genotype T3SS.train_eval.genotype \
    --phenotype train_eval/T3SS.train_eval.phenotype \
    --weights \
    --out T3SS.pkl
```

This will create a new model artifact `T3SS.pkl` in the current directory, and a tabular file `T3SS.pkl.rank` representing the relative impact of input features on prediction output as learned by the model.

3.3 Performance Estimation of Phenotrex Classifiers

The default way for phenotrex to estimate model performance (other than applying the trained model to a held back test set) is nested cross-validation (CV). This allows the estimation of predictive performance for a given set of training data and hyperparameters.

To estimate performance of the model trained in the previous section, we perform a 10x/5x nested cross-validation like so:

```
$ phenotrex cv xgb \
    --genotype T3SS.train_eval.genotype \
    --phenotype train_eval/T3SS.train_eval.phenotype \
    --out T3SS.misclassifications.tsv \
    --folds 5 \
    --replicates 10 \
    --threads 4
```

After training, predictive performance metrics averaged over outer CV folds will be printed to stderr, and a new tabular file `T3SS.misclassifications.tsv` will be created. This file contains the identifiers, phenotypic trait labels and fractions of misclassifications of the sample over outer CV folds.

Note: The above command does not accept a trained model artifact. Since cross-validation is performed by training several models on subsets of the given data, a final model is not warranted here. In general, training of the final classifier with `phenotrex train {xgb,svm}` should be performed only when satisified with performance of the selected hyperparameters as given by cross-validation.

3.4 Performance Estimation for Metagenomic Phenotrex Classifiers

For phenotrex models intended to be applied to metagenome assembled genomes, it is useful to estimate the impact of missing and/or contaminating genomic features on the model output. In phenotrex, this is achieved by randomly resampling the features of validation genomes to simulate incompleteness and contamination (see Feldbauer et al. 2015). For example, to estimate performance of a model on 80% complete and 5% contaminated genomic bins, nested cross-validation is performed where from each validation sample 20% of eggNOG cluster features are randomly removed. To simulate 5% contamination conservatively, a requisite number of eggNOG clusters are added to the genome drawn randomly only from genomes of the opposite label. This is performed at regular intervals of completeness and contamination, resulting in a JSON file detailing the estimated predictive performance at each step. By default, a grid of 5% increments of completeness and contamination is evaluated.

To perform cross-validation under consideration of contamination and completeness (CCCV), perform:

```
$ phenotrex cccv xgb \
    --genotype T3SS.train_eval.genotype \
    --phenotype train_eval/T3SS.train_eval.phenotype \
    --out T3SS.cccv.json \
    --folds 5 \
    --replicates 10 \
    --threads 4 \
    --verb
```

The above command results in a file `T3SS.cccv.json` being created, performance metrics at each step of the completeness/contamination grid.

Note: The default binary classification performance metric used by phenotrex is Balanced Accuracy (bACC), which is the arithmetic mean of Sensitivity and Specificity of prediction:

$$bACC = \frac{1}{2}(\frac{TP}{TP+FN} + \frac{TN}{TN+FP})$$

This metric avoids inflating performance estimates on imbalanced datasets, and ranges from 0.5 (performance is indistinguishable from random) to 1.0 (perfect performance).

Users are encouraged to determine the contamination and completeness levels of input metagenomic bins (e.g. using CheckM), and critically examine the validity of predictions made by the classifier using the estimated performance at the closest point in the completeness/contamination grid.

3.5 Predicting Phenotypes with Phenotrex

For prediction, the threshold confidence of the classifier can be specified - all predictions with confidence below this threshold are then masked with 'N/A'.

Note: The reported `Confidence` of the classifier is the model's internal confidence in its prediction given its input data. If the input genome is significantly incomplete or contaminated, this measure may be misleading, as the genome could be missing vital information required for correct classification by the model. For such cases, the external confidence measure for the given completeness/contamination level as computed by `phenotrex cccv {xgb,svm}` should be considered as well.

Prediction of phenotypic traits with a pre-computed genotype file derived from genomes in the `test/genomes` directory (see section *Creation of Phenotrex Input Features*):

```
$ phenotrex predict \
    --genotype test/T3SS.test.genotype \
    --classifier T3SS.pkl \
    --min_proba 0.6 \
    --verb > T3SS.test_predictions.tsv
```

The `predict` command outputs prediction results directly to stdout. When redirecting stdout to a file, this results in a 3-column TSV file of the following shape:

```
# Trait: T3SS
Identifier  Trait present   Confidence
```

(continues on next page)

(continued from previous page)

```
GCF_000006645.1.fna.gz      YES     0.8604
GCF_000006665.1.fna.gz      YES     0.8675
GCF_000006825.1.fna.gz      NO      0.6617
GCF_000007165.1.fna.gz      YES     0.6771
GCF_000007205.1.fna.gz      YES     0.8261
GCF_000007445.1.fna.gz      YES     0.8183
...
```

Lines starting with # represent metadata, in this case the trait name saved in the used model artifact.

3.6 Explanation of Phenotrex Predictions

In addition to providing predicted trait labels and confidence measures, phenotrex can provide additional explanations of its decision process. This can help debug faulty hyperparameter configurations and help identify errors in the training data. Model explanation is done by gauging the importance of input features identified in genomes at training and prediction time.

3.6.1 Feature Importance at Training Time

The relative impact of features learned by phenotrex models is output at training time when the flag `--weights` is added to the `phenotrex train {xgb,svm}` command. The meaning of the importance differs depending on the selected ML algorithm: when using XGB, the measure represents the overall importance of that feature in the decision process of the model (irrespective of the final prediction), when using SVM, the measure correlates with the probability of calling YES (positive values) or NO (negative values) for the trait in question.

3.6.2 Feature Importance at Prediction Time

A second, and arguably more useful type of explanation can be computed at prediction time. For each predicted genome, a list of features is created which, either by presence or absence, contributed most to the prediction output for that genome. Feature importance is represented by SHAP (SHapley Additive exPlanations) values. The sum of SHAP values of all features considered by the model is directly related to the probability of calling YES for the trait and genome in question.

Note: Feature explanation at prediction time is implemented by the shap package, which efficiently computes the required explanations for XGB models with de facto zero overhead. For SVM models however, this calculation can be extremely costly. We thus suggest that for use cases where model explainability is important, XGB should be preferred over SVM.

To create feature explanations at prediction time:

```
$ phenotrex predict \
    --genotype test/T3SS.test.genotype \
    --classifier T3SS.pkl \
    --min_proba 0.6 \
    --out_explain_summary T3SS.expl_summary.tsv \
    --out_explain_per_sample T3SS.expl_per_sample.tsv \
    --n_max_explained_features 10 \
    --verb > T3SS.test_predictions.tsv
```

In addition to the original output file containing predictions, two additional files have been created:

- **T3SS.expl_per_sample.tsv** This file contains for each predicted genome, the features which had the highest impact on the model output, as well as the sign of that impact.

```
rank    Sample  Feature Feature Presence        SHAP Value (class=YES)  Feature␣
↪Annotation
...
0       GCF_000006825.1.fna.gz  COG4789 0.0     -0.46379        Type iii␣
↪secretion
1       GCF_000006825.1.fna.gz  COG1025 0.0     -0.19678        Belongs to the␣
↪peptidase M16 family
2       GCF_000006825.1.fna.gz  COG0814 1.0     0.16128 amino acid
3       GCF_000006825.1.fna.gz  COG1330 1.0     0.15993 A helicase nuclease␣
↪that prepares dsDNA breaks (DSB)...
4       GCF_000006825.1.fna.gz  COG1459 1.0     0.14634 type II secretion system
5       GCF_000006825.1.fna.gz  COG1450 0.0     -0.14371        Type ii and iii␣
↪secretion system protein
...
```

  For example, for the genome `GCF_000006825.1.fna.gz`, we see that the absence of `COG4789` is the single most impactful contribution to the prediction output towards the (correct) prediction `NO`. We can immediately identify another secretory system component absent from the genome (`COG1450`) which contributes to this prediction output. However, as the used model was trained on a small toy dataset, the presence of `COG0814` with the somewhat unhelpful annotation "amino acid" and other features make significant contributions towards flipping the prediction to `YES`, leading ultimately to a correct output but with a low confidence of 0.66.

- **T3SS.expl_summary.tsv** This file contains the overall highest impact features, averaged over all SHAP contributions in all predicted genomes. For each feature, the average SHAP value change upon presence or absence of the feature is given, as well as the number of samples in which the feature was present and absent.

```
Feature Mean SHAP If Present    Mean SHAP If Absent     N(present)      ␣
↪N(absent)      Feature Annotation
COG4789 0.69559 -0.48636        29      162     Type iii secretion
COG1025 0.26914 -0.17944        46      145     Belongs to the peptidase M16␣
↪family
COG1330 0.10883 -0.12163        72      119     A helicase nuclease that␣
↪prepares dsDNA breaks (DSB)...
COG1929 0.22469 -0.08981        37      154     Belongs to the glycerate kinase␣
↪type-1 family
COG0833 0.20413 -0.08887        38      153     amino acid
COG0814 0.13396 -0.07835        60      131     amino acid
COG3835 0.18331 -0.05811        38      153     regulator
COG1459 0.11474 -0.05503        73      118     type II secretion system
COG1450 0.03356 -0.10312        107     84      Type ii and iii secretion␣
↪system protein
```

CHAPTER

FOUR

PHENOTREX

4.1 phenotrex package

4.1.1 Subpackages

phenotrex.cli package

Submodules

phenotrex.cli.cccv module

phenotrex.cli.clf_opt module

phenotrex.cli.compute_genotype module

phenotrex.cli.cv module

phenotrex.cli.generic_func module

phenotrex.cli.generic_opt module

phenotrex.cli.get_weights module

phenotrex.cli.main module

phenotrex.cli.plot module

phenotrex.cli.predict module

phenotrex.cli.train module

Module contents

phenotrex.io package

Submodules

phenotrex.io.flat module

phenotrex.io.serialization module

Module contents

phenotrex.ml package

Subpackages

phenotrex.ml.clf package

Submodules

phenotrex.ml.clf.svm module

phenotrex.ml.clf.xgbm module

Module contents

Submodules

phenotrex.ml.cccv module

phenotrex.ml.feature_select module

phenotrex.ml.trex_classifier module

phenotrex.ml.vectorizer module

Module contents

phenotrex.structure package

Submodules

phenotrex.structure.records module

class phenotrex.structure.records.**GenotypeRecord**(*identifier: str*, *feature_type: str*, *features: List[str]*)

Bases: object

Genomic features of a sample referenced by *identifier*.

feature_type: str

features: List[str]

identifier: str

class phenotrex.structure.records.**GroupRecord**(*identifier: str*, *group_name: Optional[str]*, *group_id: Optional[int]*)

Bases: object

Group label of sample *identifier*. Notes —— Useful for leave-one-group-out cross-validation (LOGO-CV), for example, to take taxonomy into account.

group_id: Optional[int]

group_name: Optional[str]

identifier: str

class phenotrex.structure.records.**PhenotypeRecord**(*identifier: str*, *trait_name: str*, *trait_sign: int*)

Bases: object

Ground truth labels of sample *identifier*, indicating presence/absence of trait *trait_name*:

- 0 if trait is absent
- 1 if trait is present

identifier: str

trait_name: str

trait_sign: int

class phenotrex.structure.records.**TrainingRecord**(*identifier: str*, *group_name: Optional[str]*, *group_id: Optional[int]*, *trait_name: str*, *trait_sign: int*, *feature_type: str*, *features: List[str]*)

Bases: *phenotrex.structure.records.GenotypeRecord*, *phenotrex.structure.records.PhenotypeRecord*, *phenotrex.structure.records.GroupRecord*

Sample containing Genotype-, Phenotype- and GroupRecords, suitable as machine learning input for a single observation.

feature_type: str

features: List[str]

identifier: str

Module contents

phenotrex.transforms package

Submodules

phenotrex.transforms.annotation module

phenotrex.transforms.resampling module

class phenotrex.transforms.resampling.**TrainingRecordResampler**(*random_state: Optional[float] = None*, *verb: bool = False*)

Bases: object

Instantiates an object which can generate versions of a TrainingRecord resampled to defined completeness and contamination levels. Requires prior fitting with full List[TrainingRecord] to get sources of contamination for both classes.

Parameters

- **random_state** – Randomness seed to use while resampling
- **verb** – Toggle verbosity

fit(*records: List[*phenotrex.structure.records.TrainingRecord*]*)

Fit TrainingRecordResampler on full TrainingRecord list to determine set of positive and negative features for contamination resampling.

Parameters **records** – the full List[TrainingRecord] on which ml training will commence.

Returns True if fitting was performed, else False.

get_resampled(*record:* phenotrex.structure.records.TrainingRecord, *comple: float = 1.0*, *conta: float = 0.0*) → *phenotrex.structure.records.TrainingRecord*

Resample a TrainingRecord to defined completeness and contamination levels. Comple=1, Conta=1 will double set size.

Parameters

- **comple** – completeness of returned TrainingRecord features. Range: 0 - 1
- **conta** – contamination of returned TrainingRecord features. Range: 0 - 1
- **record** – the input TrainingRecord

Returns a resampled TrainingRecord.

Module contents

phenotrex.transforms.**fastas_to_grs**(**args*, ***kwargs*)

phenotrex.util package

Submodules

phenotrex.util.helpers module

phenotrex.util.helpers.**fail_missing_dependency**(**args*, ***kwargs*)

phenotrex.util.helpers.**get_groups**(*records: List[*phenotrex.structure.records.TrainingRecord*]*) → numpy.ndarray

Get groups from list of TrainingRecords

Parameters **records** –

Returns list for groups

phenotrex.util.helpers.**get_x_y_tn_ft**(*records: List[*phenotrex.structure.records.TrainingRecord*]*) → Tuple[numpy.ndarray, numpy.ndarray, str, str]

Get separate X and y from list of TrainingRecord. Also infer trait name from first TrainingRecord.

Parameters **records** – a List[TrainingRecord]

Returns separate lists of features and targets, the trait name and the feature type

phenotrex.util.logging module

`phenotrex.util.logging.get_logger(initname, verb=False)`

This function provides a logger to all scripts used in this project.

Parameters

- **initname** – The name of the logger to show up in log.
- **verb** – Toggle verbosity

Returns the finished Logger object.

phenotrex.util.plotting module

phenotrex.util.taxonomy module

Module contents

4.1.2 Module contents

Top-level package for phenotrex.

CHAPTER

FIVE

CREDITS

5.1 Development Lead

- Lukas Lüftinger <lukas.lueftinger@outlook.com>

5.2 Contributors

- Patrick Hyden <hydenp89@univie.ac.at>
- Roman Feldbauer <roman.feldbauer@univie.ac.at>

CHAPTER

SIX

HISTORY

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

phenotrex, 15
phenotrex.cli, 11
phenotrex.ml.clf, 12
phenotrex.structure, 13
phenotrex.structure.records, 12
phenotrex.transforms, 14
phenotrex.transforms.resampling, 13
phenotrex.util, 15
phenotrex.util.helpers, 14
phenotrex.util.logging, 15

INDEX

F

fail_missing_dependency() (*in module phenotrex.util.helpers*), 14
fastas_to_grs() (*in module phenotrex.transforms*), 14
feature_type (*phenotrex.structure.records.GenotypeRecord attribute*), 12
feature_type (*phenotrex.structure.records.TrainingRecord attribute*), 13
features (*phenotrex.structure.records.GenotypeRecord attribute*), 12
features (*phenotrex.structure.records.TrainingRecord attribute*), 13
fit() (*phenotrex.transforms.resampling.TrainingRecordResampler method*), 14

G

GenotypeRecord (*class in phenotrex.structure.records*), 12
get_groups() (*in module phenotrex.util.helpers*), 14
get_logger() (*in module phenotrex.util.logging*), 15
get_resampled() (*phenotrex.transforms.resampling.TrainingRecordResampler method*), 14
get_x_y_tn_ft() (*in module phenotrex.util.helpers*), 14
group_id (*phenotrex.structure.records.GroupRecord attribute*), 13
group_name (*phenotrex.structure.records.GroupRecord attribute*), 13
GroupRecord (*class in phenotrex.structure.records*), 13

I

identifier (*phenotrex.structure.records.GenotypeRecord attribute*), 12
identifier (*phenotrex.structure.records.GroupRecord attribute*), 13
identifier (*phenotrex.structure.records.PhenotypeRecord attribute*), 13
identifier (*phenotrex.structure.records.TrainingRecord attribute*), 13

M

module
phenotrex, 15
phenotrex.cli, 11
phenotrex.ml.clf, 12
phenotrex.structure, 13
phenotrex.structure.records, 12
phenotrex.transforms, 14
phenotrex.transforms.resampling, 13
phenotrex.util, 15
phenotrex.util.helpers, 14
phenotrex.util.logging, 15

P

phenotrex
module, 15
phenotrex.cli
module, 11
phenotrex.ml.clf
module, 12
phenotrex.structure
module, 13
phenotrex.structure.records
module, 12
phenotrex.transforms
module, 14
phenotrex.transforms.resampling
module, 13
phenotrex.util
module, 15
phenotrex.util.helpers
module, 14
phenotrex.util.logging
module, 15
PhenotypeRecord (*class in phenotrex.structure.records*), 13

T

TrainingRecord (*class in phenotrex.structure.records*), 13
TrainingRecordResampler (*class in phenotrex.transforms.resampling*), 13
trait_name (*phenotrex.structure.records.PhenotypeRecord attribute*), 13

`trait_sign`(*phenotrex.structure.records.PhenotypeRecord attribute*), 13