SparkLightAutoML

AI Lab ML Tools

Jul 13, 2023

PYTHON API

1 sparklightautoml.automl 3

2 sparklightautoml.dataset 13

3 sparklightautoml.ml_algo 19

4 sparklightautoml.pipelines 25

5 sparklightautoml.pipelines.selection 27

6 sparklightautoml.pipelines.features 29

7 sparklightautoml.pipelines.ml 37

8 sparklightautoml.reader 39

9 sparklightautoml.report 45

10 sparklightautoml.tasks 47

11 sparklightautoml.transformers 49

12 sparklightautoml.validation 65

13 Running spark lama app on Spark YARN with spark_submit 67

14 Running spark lama app on Spark YARN 69

15 Running spark lama app on standalone cluster 79

16 Deploy on Minikube 81

17 Environment setup 83

18 Run examples in minikube 85

19 Running spark lama app on Kubernetes cluster 87

Index 89

This is a distributed version of LAMA library written on Spark framework. SLAMA brings LAMA functionality on Spark including:

- Automatic hyperparameter tuning, data processing.
- Automatic typing, feature selection.
- Automatic time utilization.
- Automatic report creation.
- Easy-to-use modular scheme to create your own pipelines.
- Support of Spark ML pipelines, including saving/loading.
- Caching and checkpointing of intermediate results

Known limitations: - Only the tabular preset is currently supported

CHAPTER

ONE

SPARKLIGHTAUTOML.AUTOML

The main module, which includes the SparkAutoML class, blenders and ready-made presets.

SparkAutoML	Class for compile full pipeline of AutoML task.

1.1 SparkAutoML

class sparklightautoml.automl.base.**SparkAutoML**(*reader=None*, *levels=None*, *timer=None*, *blender=None*, *skip_conn=False*, *return_all_predictions=False*, *computation_settings=('no_parallelism', -1)*)

Bases: `TransformerInputOutputRoles`

Class for compile full pipeline of AutoML task.

AutoML steps:

- Read, analyze data and get inner `LAMLDataset` from input dataset: performed by reader.
- Create validation scheme.
- Compute passed ml pipelines from levels. Each element of levels is list of `MLPipelines` prediction from current level are passed to next level pipelines as features.
- Time monitoring - check if we have enough time to calc new pipeline.
- Blend last level models and prune useless pipelines to speedup inference: performed by blender.
- Returns prediction on validation data. If crossvalidation scheme is used, out-of-fold prediction will returned. If validation data is passed it will return prediction on validation dataset. In case of cv scheme when some point of train data never was used as validation (ex. timeout exceeded or custom cv iterator like `TimeSeriesIterator` was used) NaN for this point will be returned.

Example

Common usecase - create custom pipelines or presets.

```
>>> reader = SparkToSparkReader()
>>> pipe = SparkMLPipeline([SparkMLAlgo()])
>>> levels = [[pipe]]
>>> automl = SparkAutoML(reader, levels, )
>>> automl.fit_predict(data, roles={'target': 'TARGET'})
```

__init__(*reader=None*, *levels=None*, *timer=None*, *blender=None*, *skip_conn=False*, *return_all_predictions=False*, *computation_settings=('no_parallelism', -1)*)

Parameters

- **reader** (Optional[*SparkToSparkReader*]) – Instance of Reader class object that creates LAMLDataset from input data.
- **levels** (Optional[Sequence[Sequence[*SparkMLPipeline*]]]) – List of list of MLPipelines.
- **timer** (Optional[PipelineTimer]) – Timer instance of PipelineTimer. Default - unlimited timer.
- **blender** (Optional[*SparkBlender*]) – Instance of Blender. Default - BestModelSelector.
- **skip_conn** (bool) – True if we should pass first level input features to next levels.

Note: There are several verbosity levels:

- *0*: No messages.
- *1*: Warnings.
- *2*: Info.
- *3*: Debug.

---

fit_predict(*train_data*, *roles*, *train_features=None*, *cv_iter=None*, *valid_data=None*, *valid_features=None*, *verbose=0*, *persistence_manager=None*)

Fit on input data and make prediction on validation part.

Parameters

- **train_data** (Any) – Dataset to train.
- **roles** (dict) – Roles dict.
- **train_features** (Optional[Sequence[str]]) – Optional features names, if cannot be inferred from train_data.
- **cv_iter** (Optional[Iterable]) – Custom cv iterator. For example, TimeSeriesIterator.
- **valid_data** (Optional[Any]) – Optional validation dataset.
- **valid_features** (Optional[Sequence[str]]) – Optional validation dataset features if can't be inferred from *valid_data*.
- **verbose** (int) – controls verbosity

Return type
SparkDataset

Returns
Predicted values.

predict(*data*, *return_all_predictions=None*, *add_reader_attrs=False*, *persistence_manager=None*)

Get dataset with predictions.

Almost same as lightautoml.automl.base.AutoML.predict on new dataset, with additional features.

Additional features - working with different data formats. Supported now:

- Path to `.csv`, `.parquet`, `.feather` files.
- `ndarray`, or dict of `ndarray`. For example, `{'data': X...}`. In this case roles are optional, but *train_features* and *valid_features* required.
- `pandas.DataFrame`.

Parallel inference - you can pass `n_jobs` to speedup prediction (requires more RAM). Batch_inference - you can pass `batch_size` to decrease RAM usage (may be longer).

Parameters

- **data** (`Union[str, DataFrame]`) – Dataset to perform inference.
- **features_names** – Optional features names, if cannot be inferred from *train_data*.
- **return_all_predictions** (`Optional[bool]`) – if True, returns all model predictions from last level
- **add_reader_attrs** (`bool`) – if True, the reader's attributes will be added to the SparkDataset

Return type
SparkDataset

Returns
Dataset with predictions.

`collect_used_feats()`

Get feats that automl uses on inference.

Return type
`List[str]`

Returns
Features names list.

`collect_model_stats()`

Collect info about models in automl.

Return type
`Dict[str, int]`

Returns
Dict with models and its runtime numbers.

1.2 Presets

Presets for end-to-end model training for special tasks.

base.SparkAutoMLPreset	Basic class for automl preset.
tabular_presets.SparkTabularAutoML	Spark version of `TabularAutoML`.

1.2.1 SparkAutoMLPreset

class sparklightautoml.automl.presets.base.**SparkAutoMLPreset**(*task*, *timeout=3600*, *memory_limit=16*, *cpu_limit=4*, *gpu_ids='all'*, *timing_params=None*, *config_path=None*, *computation_settings=('no_parallelism', -1)*, ***kwargs*)

Bases: *SparkAutoML*

Basic class for automl preset.

It's almost like AutoML, but with delayed initialization. Initialization starts on fit, some params are inferred from data. Preset should be defined via `.create_automl` method. Params should be set via yaml config. Most usefull case - end-to-end model development.

Example

```
>>> automl = SparkAutoMLPreset(SparkTask('binary'), timeout=3600)
>>> automl.fit_predict(data, roles={'target': 'TARGET'})
```

__init__(*task*, *timeout=3600*, *memory_limit=16*, *cpu_limit=4*, *gpu_ids='all'*, *timing_params=None*, *config_path=None*, *computation_settings=('no_parallelism', -1)*, ***kwargs*)

Commonly _params kwargs (ex. timing_params) set via config file (config_path argument). If you need to change just few params, it's possible to pass it as dict of dicts, like json. To get available params please look on default config template. Also you can find there param description. To generate config template call `SomePreset.get_config('config_path.yml')`.

Parameters

- **task** (*SparkTask*) – Task to solve.
- **timeout** (int) – Timeout in seconds.
- **memory_limit** (int) – Memory limit that are passed to each automl.
- **cpu_limit** (int) – CPU limit that that are passed to each automl.
- **gpu_ids** (Optional[str]) – GPU IDs that are passed to each automl.
- **verbose** – Controls the verbosity: the higher, the more messages. <1 : messages are not displayed; >=1 : the computation process for layers is displayed; >=2 : the information about folds processing is also displayed; >=3 : the hyperparameters optimization process is also displayed; >=4 : the training process for every algorithm is displayed;
- **timing_params** (Optional[dict]) – Timing param dict.
- **config_path** (Optional[str]) – Path to config file.
- ****kwargs** (Any) – Not used.

classmethod get_config(*path=None*)

Create new config template.

Parameters

path (Optional[str]) – Path to config.

Return type

Optional[dict]

Returns
Config.

`create_automl`(***fit_args*)

Abstract method - how to build automl.

Here you should create all automl components, like readers, levels, timers, blenders. Method `._initialize` should be called in the end to create automl.

Parameters
`**fit_args` – params that are passed to `.fit_predict` method.

`fit_predict`(*train_data*, *roles*, *train_features=None*, *cv_iter=None*, *valid_data=None*, *valid_features=None*, *verbose=0*, *persistence_manager=None*)

Fit on input data and make prediction on validation part.

Parameters

- **`train_data`** (`Any`) – Dataset to train.
- **`roles`** (`dict`) – Roles dict.
- **`train_features`** (`Optional[Sequence[str]]`) – Features names, if can't be inferred from *train_data*.
- **`cv_iter`** (`Optional[Iterable]`) – Custom cv-iterator. For example, `TimeSeriesIterator`.
- **`valid_data`** (`Optional[Any]`) – Optional validation dataset.
- **`valid_features`** (`Optional[Sequence[str]]`) – Optional validation dataset features if can't be inferred from *valid_data*.
- **`verbose`** (`int`) – Verbosity level that are passed to each automl.

Return type
SparkDataset

Returns
Dataset with predictions. Call `.data` to get predictions array.

`static set_verbosity_level`(*verbose*)

Verbosity level setter.

Parameters
`verbose` (`int`) – Controls the verbosity: the higher, the more messages. <1 : messages are not displayed; >=1 : the computation process for layers is displayed; >=2 : the information about folds processing is also displayed; >=3 : the hyperparameters optimization process is also displayed; >=4 : the training process for every algorithm is displayed;

1.2.2 SparkTabularAutoML

class sparklightautoml.automl.presets.tabular_presets.**SparkTabularAutoML**(*spark*, *task*, *timeout=3600*, *memory_limit=16*, *cpu_limit=4*, *gpu_ids='all'*, *timing_params=None*, *config_path=None*, *general_params=None*, *reader_params=None*, *read_csv_params=None*, *nested_cv_params=None*, *tuning_params=None*, *selection_params=None*, *lgb_params=None*, *cb_params=None*, *linear_l2_params=None*, *gbm_pipeline_params=None*, *linear_pipeline_params=None*, *persistence_manager=None*, *computation_settings=('no_parallelism', -1)*)

Bases: *SparkAutoMLPreset*

Spark version of `TabularAutoML`. Represent high level entity of spark lightautoml. Use this class to create automl instance.

Example

```
>>> automl = SparkTabularAutoML(
>>>     spark=spark,
>>>     task=SparkTask('binary'),
>>>     general_params={"use_algos": [["lgb"]]},
>>>     lgb_params={'use_single_dataset_mode': True},
>>>     reader_params={"cv": cv, "advanced_roles": False}
>>> )
>>> oof_predictions = automl.fit_predict(
>>>     train_data,
>>>     roles=roles
>>> )
```

---

create_automl(***fit_args*)

Create basic automl instance.

Parameters

****fit_args** – Contain all information needed for creating automl.

fit_predict(*train_data*, *roles=None*, *train_features=None*, *cv_iter=None*, *valid_data=None*, *valid_features=None*, *log_file=None*, *verbose=0*, *persistence_manager=None*)

Fit and get prediction on validation dataset.

Almost same as `lightautoml.automl.base.AutoML.fit_predict`.

Additional features - working with different data formats. Supported now:

- Path to `.csv`, `.parquet`, `.feather` files.
- `ndarray`, or dict of `ndarray`. For example, `{'data': X...}`. In this case, roles are optional, but *train_features* and *valid_features* required.
- `pandas.DataFrame`.

Parameters

- **train_data** (`Union[str, DataFrame]`) – Dataset to train.
- **roles** (`Optional[dict]`) – Roles dict.
- **train_features** (`Optional[Sequence[str]]`) – Optional features names, if can't be inferred from *train_data*.
- **cv_iter** (`Optional[Iterable]`) – Custom cv-iterator. For example, `TimeSeriesIterator`.
- **valid_data** (`Union[str, DataFrame, None]`) – Optional validation dataset.
- **valid_features** (`Optional[Sequence[str]]`) – Optional validation dataset features if cannot be inferred from *valid_data*.
- **verbose** (`int`) – Controls the verbosity: the higher, the more messages. <1 : messages are not displayed; >=1 : the computation process for layers is displayed; >=2 : the information about folds processing is also displayed; >=3 : the hyperparameters optimization process is also displayed; >=4 : the training process for every algorithm is displayed;
- **log_file** (`Optional[str]`) – Filename for writing logging messages. If log_file is specified,
- **exists** (*the messages will be saved in a the file. If the file*) –
- **overwritten.** (*it will be*) –

Return type
SparkDataset

Returns
Dataset with predictions. Call `.data` to get predictions array.

static get_pdp_data_numeric_feature(*df*, *feature_name*, *model*, *prediction_col*, *n_bins*, *ice_fraction=1.0*, *ice_fraction_seed=42*)

Returns *grid*, *ys* and *counts* calculated on input numeric column to plot PDP.

Parameters

- **df** (*SparkDataFrame*) – Spark DataFrame with *feature_name* column
- **feature_name** (*str*) – feature column name
- **model** (*PipelineModel*) – Spark Pipeline Model
- **prediction_col** (*str*) – prediction column to be created by the *model*
- **n_bins** (*int*) – The number of bins to produce. Raises exception if n_bins < 2.

- **ice_fraction** (*float, optional*) – What fraction of the input dataframe will be used to make predictions. Useful for very large dataframe. Defaults to 1.0.
- **ice_fraction_seed** (*int, optional*) – Seed for *ice_fraction*. Defaults to 42.

Returns
grid is list of categories, *ys* is list of predictions by category, *counts* is numbers of values by category

Return type
Tuple[List, List, List]

static get_pdp_data_categorical_feature(*df*, *feature_name*, *model*, *prediction_col*, *n_top_cats*, *ice_fraction=1.0*, *ice_fraction_seed=42*)

Returns *grid*, *ys* and *counts* calculated on input categorical column to plot PDP.

Parameters

- **df** (*SparkDataFrame*) – Spark DataFrame with *feature_name* column
- **feature_name** (*str*) – feature column name
- **model** (*PipelineModel*) – Spark Pipeline Model
- **prediction_col** (*str*) – prediction column to be created by the *model*
- **n_top_cats** (*int*) – param to selection top n categories
- **ice_fraction** (*float, optional*) – What fraction of the input dataframe will be used to make predictions. Useful for very large dataframe. Defaults to 1.0.
- **ice_fraction_seed** (*int, optional*) – Seed for *ice_fraction*. Defaults to 42.

Returns
grid is list of categories, *ys* is list of predictions by category, *counts* is numbers of values by category

Return type
Tuple[List, List, List]

static get_pdp_data_datetime_feature(*df*, *feature_name*, *model*, *prediction_col*, *datetime_level*, *reader*, *ice_fraction=1.0*, *ice_fraction_seed=42*)

Returns *grid*, *ys* and *counts* calculated on input datetime column to plot PDP.

Parameters

- **df** (*SparkDataFrame*) – Spark DataFrame with *feature_name* column
- **feature_name** (*str*) – feature column name
- **model** (*PipelineModel*) – Spark Pipeline Model
- **prediction_col** (*str*) – prediction column to be created by the *model*
- **datetime_level** (*str*) – Unit of time that will be modified to calculate dependence: “year”, “month” or “dayofweek”
- **reader** (*_type_*) – Automl reader to transform input dataframe before *model* inferring.
- **ice_fraction** (*float, optional*) – What fraction of the input dataframe will be used to make predictions. Useful for very large dataframe. Defaults to 1.0.
- **ice_fraction_seed** (*int, optional*) – Seed for *ice_fraction*. Defaults to 42.

Returns
grid is list of categories, *ys* is list of predictions by category, *counts* is numbers of values by category

Return type
Tuple[List, List, List]

1.3 Blenders

SparkBlender	Basic class for blending.
SparkBestModelSelector	Select best single model from level.
SparkMeanBlender	Simple average level predictions.
SparkWeightedBlender	Weighted Blender based on coord descent, optimize task metric directly.

1.3.1 SparkBlender

class sparklightautoml.automl.blend.**SparkBlender**

Bases: TransformerInputOutputRoles, ABC

Basic class for blending.

Blender learns how to make blend on sequence of prediction datasets and prune pipes, that are not used in final blend.

property input_roles

Returns dict of input roles

property output_roles

Returns dict of output roles

transformer(**args*, ***kwargs*)

Returns Spark MLlib Transformer. Represents a Transformer with fitted models.

Return type
Optional[Transformer]

split_models(*predictions*, *pipes*)

Split predictions by single model prediction datasets.

Parameters

- **predictions** (*SparkDataset*) – Dataset with predictions.
- **pipes** (Sequence[*SparkMLPipeline*]) – ml pipelines to be associated with the predictions

Returns

- prediction column name
- corresponding model index (in the pipe)
- corresponding pipe index

Return type
Each tuple in the list is

score(*dataset*)

Score metric for blender.

Parameters
dataset (*SparkDataset*) – Blended predictions dataset.

Return type
float

Returns
Metric value.

1.3.2 SparkBestModelSelector

class sparklightautoml.automl.blend.**SparkBestModelSelector**

Bases: *SparkBlender*, WeightedBlender

Select best single model from level.

Drops pipes that are not used in calc best model. Works in general case (even on some custom things) and most efficient on inference. Perform worse than other on tables, specially if some of models was terminated by timer.

1.3.3 SparkMeanBlender

class sparklightautoml.automl.blend.**SparkMeanBlender**

Bases: *SparkBlender*

Simple average level predictions.

Works only with TabularDatasets. Doesn't require target to fit. No pruning.

1.3.4 SparkWeightedBlender

class sparklightautoml.automl.blend.**SparkWeightedBlender**(*max_iters=5*, *max_inner_iters=7*, *max_nonzero_coef=0.05*)

Bases: *SparkBlender*, WeightedBlender

Weighted Blender based on coord descent, optimize task metric directly.

Weight sum eq. 1. Good blender for tabular data, even if some predictions are NaN (ex. timeout). Model with low weights will be pruned.

CHAPTER

TWO

SPARKLIGHTAUTOML.DATASET

Provides base entities for working with data.

2.1 Dataset Interfaces

base.SparkDataset	Implements a dataset that uses a `pyspark.sql.DataFrame` internally, stores some internal state (features, roles, ...) and provide methods to work with dataset.
base.PersistenceLevel	Used for signaling types of persistence points encountered during AutoML process.
base.PersistenceManager	Base interface of an entity responsible for caching and storing intermediate results somewhere.
base.Unpersistable	Interface to provide for external entities to unpersist dataframes and files stored by the entity that implements this interface

2.1.1 SparkDataset

class sparklightautoml.dataset.base.**SparkDataset**(*data*, *roles*, *persistence_manager=None*, *task=None*, *bucketized=False*, *dependencies=None*, *name=None*, *target=None*, *folds=None*, ***kwargs*)

Bases: `LAMLDataset`, *Unpersistable*

Implements a dataset that uses a `pyspark.sql.DataFrame` internally, stores some internal state (features, roles, ...) and provide methods to work with dataset.

classmethod concatenate(*datasets*, *name=None*, *extra_dependencies=None*)

Concat multiple datasets by joining their internal `pyspark.sql.DataFrame` using inner join on special hidden '_id' column :type datasets: `Sequence`[*SparkDataset*] :param datasets: spark datasets to be joined

Return type
SparkDataset

Returns
a joined dataset, containing features (and columns too) from all datasets except containing only one _id column

property features

Get list of features.

Returns
list of features.

property roles

Roles dict.

set_data(*data*, *features*, *roles=None*, *persistence_manager=None*, *dependencies=None*, *uid=None*, *name=None*, *frozen=False*)

Inplace set data, features, roles for empty dataset.

Parameters

- **data** (DataFrame) – Table with features.
- **features** (List[str]) – `ignored, always None. just for same interface.
- **roles** (Union[Sequence[ColumnRole], ColumnRole, Dict[str, ColumnRole], None]) – Dict with roles.

persist(*level=None*, *force=False*)

Materializes current Spark DataFrame and unpersists all its dependencies :type level: Optional[*PersistenceLevel*] :param level:

Return type
SparkDataset

Returns
a new SparkDataset that is persisted and materialized

unpersist()

Unpersists current dataframe if it is persisted and all its dependencies. It does nothing if the dataset is frozen

2.1.2 PersistenceLevel

class sparklightautoml.dataset.base.**PersistenceLevel**(*value*)

Bases: Enum

Used for signaling types of persistence points encountered during AutoML process.

2.1.3 PersistenceManager

class sparklightautoml.dataset.base.**PersistenceManager**

Bases: ABC

Base interface of an entity responsible for caching and storing intermediate results somewhere.

abstract property all_datasets

Returns: all persisted datasets including persisted with children contexts

2.1.4 Unpersistable

class sparklightautoml.dataset.base.**Unpersistable**

Bases: ABC

Interface to provide for external entities to unpersist dataframes and files stored by the entity that implements this interface

2.2 Roles

Role contains information about the column, which determines how it is processed.

NumericVectorOrArrayRole	Role that describe numeric vector or numeric array.

2.2.1 NumericVectorOrArrayRole

class sparklightautoml.dataset.roles.**NumericVectorOrArrayRole**(*size*, *element_col_name_template*, *dtype=numpy.float32*, *force_input=False*, *prob=False*, *discretization=False*, *is_vector=True*)

Bases: NumericRole

Role that describe numeric vector or numeric array.

__init__(*size*, *element_col_name_template*, *dtype=numpy.float32*, *force_input=False*, *prob=False*, *discretization=False*, *is_vector=True*)

Parameters

- **size** (int) – number of elements in every vector in this column
- **element_col_name_template** (Union[str, List[str]]) – string template to produce name for each element in the vector
- **neccessary** (*when array-to-columns transformation is*) –
- **dtype** (Union[Callable, type, str]) – type of the vector's elements
- **force_input** (bool) – Select a feature for training, regardless of the selector results.
- **prob** (bool) – If input number is probability.

feature_name_at(*position*)

produces a name for feature on position in the vector

Parameters
position (int) – position in the vector in range [0 .. size]

Return type
str

Returns
feature name

2.3 Persistence

Persistence managers are responsible for caching and storing intermediate results on various steps during AutoML process. Storing intermediate results is required by various reasons.

Depending on the manager, it can be used for the following goals: * Support iterative data processing and preventing repeatition of calculations * Prunning of long plans that slows down catalyst optimizer * Prunning of long lineages that increase overheads on tasks serialization (and may lead to large broadcasts) * Creating reliable checkpoints preventing long recalculations in case of failures * Optimize joins converting them into shuffle-less merge joins instead of SortMerge joins (optimization of split-merge patterns in the process of multiple models/multiple feature generation)

For instance, PlainCachePersistenceManager allows to support iterative data processing and provides fast storing due to leveraging Spark caching mechanism which may employ RAM, but cannot provide for the rest of goals. From the other side, BucketedPersistenceManager can deliver for all the goals, but requires more time to store data due to writing to external storage like HDFS. LocalCheckpointPersistenceManager is in the middle: it can deliver only the first three goals, but store data fast leveraging RAM and DISK if necessary

Different persistence managers may be of more usefulness depending on current step in the automl process. There can be found several explicit levels of storing stated in PersistenceLevel entity: * READER level marks the beginning of the automl process, root of all pipelines, executed only onces. * REGULAR means storing data somewhere in the middle of ML pipeline, mainly feature processing or model training. * CHECKPOINT is used for denoting data storing in the very end of ML pipeline. These data will consist only of predictions made by one or several ML models thus making the dataframe being stored relatively small.

All persistence managers can be divided on two main types depending on how they handle different levels supplied during calling .persist(): * simple managers, that exploit the same approach to store intermediate results on all levels * composite managers (their name starts with 'Composite' prefix) that can employ different approaches to store data for different levels.

CompositeBucketedPersistenceManager should be used in most cases. It creates a bucketed dataset on READER level, which is an expensive operation executed only once. In exchanges, it leads to making all joins (on the direct descendants of the main dataframe) in the downstream process to be either broadcast joins or merge joins. In both cases it wouldn't require shuffle. On REGULAR level, mainly for the sake of supporting fast iterative data processing, it employs PlainCachePersistenceManager. On CHECKPOINT level, having a relatively small dataframe after the end of heavy data processing and computations this manager opts to reliable data storing using BucketedPersistenceManager. This choice is also motivated by prunning of long plan and lineage which have grown large up to this moment.

CompositePlainCachePersistenceManager uses PlainCachePersistenceManager for READER and REGULAR levels, avoiding expensive initial creation of a bucketed dataset. On CHECKPOINT level, it relies on BucketedPersistenceManager with the same motivation as for the previous case. However, it does have some advantages it should be used with caution. Use cases when it may be used requires specific Spark Session and AutoML configurations having the following traits: * AutoML has only one level of ML pipelines or two levels with skip_conn=False * autoBroadcastJoinThreshold is set high enough to handle some minor joins The alternative case: * AutoML has two levels with skip_conn=True * autoBroadcastJoinThreshold is set sufficiently high to make joining the main dataframe with resulting dataframes from the first level (containing predictions) shuffle-less

These conditions may change in the future.

BasePersistenceManager	Abstract implementation of base persistence functionality, including registering and de-registering what have been requested to persist/un-persist
PlainCachePersistenceManager	Manager that uses Spark .cache() / .persist() methods
LocalCheckpointPersistenceManager	Manager that uses Spark .localCheckpoint() method
BucketedPersistenceManager	Manager that uses Spark Warehouse folder to store bucketed datasets (.bucketBy .
CompositePersistenceManager	Universal composite manager that can combine other manager to apply different storing strategies on different levels.
CompositeBucketedPersistenceManager	Combines bucketing on READER and CHECKPOINT levels with PlainCache on REGULAR level.
CompositePersistenceManager	Universal composite manager that can combine other manager to apply different storing strategies on different levels.

2.3.1 BasePersistenceManager

class sparklightautoml.dataset.persistence.**BasePersistenceManager**(*parent=None*)

Bases: *PersistenceManager*

Abstract implementation of base persistence functionality, including registering and de-registering what have been requested to persist/un-persist

2.3.2 PlainCachePersistenceManager

class sparklightautoml.dataset.persistence.**PlainCachePersistenceManager**(*parent=None*, *prune_history=False*)

Bases: *BasePersistenceManager*

Manager that uses Spark .cache() / .persist() methods

2.3.3 LocalCheckpointPersistenceManager

class sparklightautoml.dataset.persistence.**LocalCheckpointPersistenceManager**(*parent=None*)

Bases: *BasePersistenceManager*

Manager that uses Spark .localCheckpoint() method

2.3.4 BucketedPersistenceManager

class sparklightautoml.dataset.persistence.**BucketedPersistenceManager**(*bucketed_datasets_folder*, *bucket_nums=100*, *parent=None*, *no_unpersisting=False*)

Bases: *BasePersistenceManager*

Manager that uses Spark Warehouse folder to store bucketed datasets (.bucketBy … .sortBy … .saveAsTable) To make such storing reliable, one should set 'spark.sql.warehouse.dir' to HDFS or other reliable storage.

2.3.5 CompositePersistenceManager

class sparklightautoml.dataset.persistence.**CompositePersistenceManager**(*level2manager*, *parent=None*)

Bases: *BasePersistenceManager*

Universal composite manager that can combine other manager to apply different storing strategies on different levels.

For BucketedPersistenceManager all unpersisting operations are delayed until the end of automl processing, due to possible loss of source for downstream persistence manager if they don't use external storage and files.

2.3.6 CompositeBucketedPersistenceManager

class sparklightautoml.dataset.persistence.**CompositeBucketedPersistenceManager**(*bucketed_datasets_folder*, *bucket_nums*)

Bases: *CompositePersistenceManager*

Combines bucketing on READER and CHECKPOINT levels with PlainCache on REGULAR level.

CHAPTER

THREE

SPARKLIGHTAUTOML.ML_ALGO

Models used for machine learning pipelines.

3.1 Base Classes

SparkTabularMLAlgo	Machine learning algorithms that accepts numpy arrays as input.
AveragingTransformer	Transformer that gets one or more columns and produce column with average values.

3.1.1 SparkTabularMLAlgo

class sparklightautoml.ml_algo.base.**SparkTabularMLAlgo**(*default_params=None*, *freeze_defaults=True*, *timer=None*, *optimization_search_space=None*, *persist_output_dataset=True*, *computations_settings=None*)

Bases: `MLAlgo`, `TransformerInputOutputRoles`, `ABC`

Machine learning algorithms that accepts numpy arrays as input.

property features

Get list of features.

fit_predict_single_fold(*fold_prediction_column*, *validation_column*, *train*, *runtime_settings=None*)

Train on train dataset and predict on holdout dataset.

Parameters

- **fold_prediction_column** (`str`) – column name for predictions made for this fold
- **validation_column** (`str`) – name of the column that signals if this row is from train or val
- **train** (*SparkDataset*) – dataset containing both train and val rows.
- **runtime_settings** (`Optional`[`Dict`[`str`, `Any`]]) – settings important for parallelism and performance that can depend on running processes
- **moment** (*at the*) –

Return type
Tuple[PipelineModel, DataFrame, str]

Returns
Target predictions for valid dataset.

3.1.2 AveragingTransformer

class sparklightautoml.ml_algo.base.**AveragingTransformer**(*task_name=None*, *input_cols=None*, *output_col='averaged_values'*, *remove_cols=None*, *convert_to_array_first=False*, *weights=None*, *dim_num=1*)

Bases: Transformer, HasInputCols, HasOutputCol, DefaultParamsWritable, DefaultParamsReadable

Transformer that gets one or more columns and produce column with average values.

__init__(*task_name=None*, *input_cols=None*, *output_col='averaged_values'*, *remove_cols=None*, *convert_to_array_first=False*, *weights=None*, *dim_num=1*)

Parameters

- **task_name** (*str, optional*) – Task name: “binary”, “multiclass” or “reg”.
- **input_cols** (*List[str], optional*) – List of input columns.
- **output_col** (*str, optional*) – Output column name. Defaults to “averaged_values”.
- **remove_cols** (*Optional[List[str]], optional*) – Columns need to remove. Defaults to None.
- **convert_to_array_first** (*bool, optional*) – If *True* then will be convert input vectors to arrays. Defaults to False.
- **weights** (*Optional[List[int]], optional*) – List of weights to scaling output values. Defaults to None.
- **dim_num** (*int, optional*) – Dimension of input columns. Defaults to 1.

3.2 Available Models

linear_pyspark.SparkLinearLBFGS	LBFGS L2 regression based on Spark MLlib.
boost_lgbm.SparkBoostLGBM	Gradient boosting on decision trees from LightGBM library.

3.2.1 SparkLinearLBFGS

class sparklightautoml.ml_algo.linear_pyspark.**SparkLinearLBFGS**(*default_params=None*, *freeze_defaults=True*, *timer=None*, *optimization_search_space=None*, *persist_output_dataset=True*, *computations_settings=None*)

Bases: *SparkTabularMLAlgo*

LBFGS L2 regression based on Spark MLlib.

default_params:

- tol: The tolerance for the stopping criteria.
- maxIter: Maximum iterations of L-BFGS.
- aggregationDepth: Param for suggested depth for treeAggregate.
- elasticNetParam: Elastic net parameter.
- regParam: Regularization parameter.
- early_stopping: Maximum rounds without improving.

freeze_defaults:

- `True` : params may be rewrited depending on dataset.
- `False`: params may be changed only manually or with tuning.

timer: `Timer` instance or `None`.

predict_single_fold(*dataset*, *model*)

Implements prediction on single fold.

Parameters

- **model** (`PipelineModel`) – Model uses to predict.
- **dataset** (*SparkDataset*) – `SparkDataset` used for prediction.

Return type
`DataFrame`

Returns
Predictions for input dataset.

3.2.2 SparkBoostLGBM

class sparklightautoml.ml_algo.boost_lgbm.**SparkBoostLGBM**(*default_params=None*, *freeze_defaults=True*, *timer=None*, *optimization_search_space=None*, *use_single_dataset_mode=True*, *max_validation_size=10000*, *chunk_size=4000000*, *convert_to_onnx=False*, *mini_batch_size=5000*, *seed=42*, *parallelism=1*, *use_barrier_execution_mode=False*, *experimental_parallel_mode=False*, *persist_output_dataset=True*, *computations_settings=None*)

Bases: *SparkTabularMLAlgo*, `ImportanceEstimator`

Gradient boosting on decision trees from LightGBM library.

default_params: All available parameters listed in synapse.ml documentation:

- https://mmlspark.blob.core.windows.net/docs/0.9.5/pyspark/synapse.ml.lightgbm.html#module-synapse.ml.lightgbm.LightGBMClassifier
- https://mmlspark.blob.core.windows.net/docs/0.9.5/pyspark/synapse.ml.lightgbm.html#module-synapse.ml.lightgbm.LightGBMRegressor

freeze_defaults:

- `True` : params may be rewritten depending on dataset.
- `False`: params may be changed only manually or with tuning.

timer: `Timer` instance or `None`.

fit_predict(*train_valid_iterator*)

Fit and then predict accordig the strategy that uses train_valid_iterator.

If item uses more then one time it will predict mean value of predictions. If the element is not used in training then the prediction will be `numpy.nan` for this item

Parameters
train_valid_iterator (*SparkBaseTrainValidIterator*) – Classic cv-iterator.

Return type
SparkDataset

Returns
Dataset with predicted values.

3.3 Utilities

boost_lgbm.LightGBMModelWrapper	Simple wrapper for *synapse.ml.lightgbm.[LightGBMRegressionModel|Light* to fix issue with loading model from saved composite pipeline.
boost_lgbm.ONNXModelWrapper	Simple wrapper for *ONNXModel* to fix issue with loading model from saved composite pipeline.

3.3.1 LightGBMModelWrapper

class sparklightautoml.ml_algo.boost_lgbm.**LightGBMModelWrapper**(*model=None*)

Bases: Transformer, MLWritable, MLReadable

Simple wrapper for *synapse.ml.lightgbm.[LightGBMRegressionModel|LightGBMClassificationModel]* to fix issue with loading model from saved composite pipeline.

For more details see: https://github.com/microsoft/SynapseML/issues/614.

classmethod read()

Returns an MLReader instance for this class.

3.3.2 ONNXModelWrapper

class sparklightautoml.ml_algo.boost_lgbm.**ONNXModelWrapper**(*model=None*)

Bases: Transformer, MLWritable, MLReadable

Simple wrapper for *ONNXModel* to fix issue with loading model from saved composite pipeline.

For more details see: https://github.com/microsoft/SynapseML/issues/614.

classmethod read()

Returns an MLReader instance for this class.

CHAPTER

FOUR

SPARKLIGHTAUTOML.PIPELINES

Pipelines for solving different tasks.

4.1 Utility traits

CHAPTER

FIVE

SPARKLIGHTAUTOML.PIPELINES.SELECTION

Feature selection module for ML pipelines.

5.1 Base Classes

SparkImportanceEstimator	

5.1.1 SparkImportanceEstimator

class sparklightautoml.pipelines.selection.base.**SparkImportanceEstimator**

Bases: ImportanceEstimator, ABC

5.2 Importance Based Selectors

permutation_importance_based.SparkNpPermutationImportanceEstimator	Permutation importance based estimator.

5.2.1 SparkNpPermutationImportanceEstimator

class sparklightautoml.pipelines.selection.permutation_importance_based.**SparkNpPermutationImportanceEst**

Bases: *SparkImportanceEstimator*

Permutation importance based estimator.

Importance calculate, using random permutation of items in single column for each feature.

__init__(*random_state=42*, *computations_settings=None*)

Parameters

random_state (int) – seed for random generation of features permutation.

fit(*train_valid=None*, *ml_algo=None*, *preds=None*)

Find importances for each feature in dataset.

Parameters

- **train_valid** (Optional[*SparkBaseTrainValidIterator*]) – Initial dataset iterator.
- **ml_algo** (Optional[*SparkTabularMLAlgo*]) – Algorithm.
- **preds** (Optional[*SparkDataset*]) – Predicted target values for validation dataset.

CHAPTER

SIX

SPARKLIGHTAUTOML.PIPELINES.FEATURES

Pipelines for features generation.

6.1 Base Classes

SparkFeaturesPipeline	Abstract class.
SparkTabularDataFeatures	Helper class contains basic features transformations for tabular data.
SparkEmptyFeaturePipeline	This class creates pipeline with SparkNoOpTransformer
SparkNoOpTransformer	This transformer does nothing, it just returns the input dataframe unchanged.
FittedPipe	

6.1.1 SparkFeaturesPipeline

class sparklightautoml.pipelines.features.base.**SparkFeaturesPipeline**(***kwargs*)

Bases: FeaturesPipeline, TransformerInputOutputRoles

Abstract class.

Analyze train dataset and create composite transformer based on subset of features. Instance can be interpreted like Transformer (look for LAMLTransformer) with delayed initialization (based on dataset metadata) Main method, user should define in custom pipeline is .create_pipeline. For example, look at LGBSimpleFeatures. After FeaturePipeline instance is created, it is used like transformer with .fit_transform and .transform method.

create_pipeline(*train*)

Analyse dataset and create composite transformer.

Parameters

train (*SparkDataset*) – Dataset with train data.

Return type

Union[*SparkBaseEstimator*, *SparkBaseTransformer*, *SparkUnionTransformer*, *SparkSequentialTransformer*]

Returns

Composite transformer (pipeline).

fit_transform(*train*)

Create pipeline and then fit on train data and then transform.

Parameters
train (*SparkDataset*) – Dataset with train data.n

Return type
SparkDataset

Returns
Dataset with new features.

6.1.2 SparkTabularDataFeatures

class sparklightautoml.pipelines.features.base.**SparkTabularDataFeatures**(***kwargs*)

Bases: object

Helper class contains basic features transformations for tabular data.

This method can de shared by all tabular feature pipelines, to simplify `.create_automl` definition.

__init__(***kwargs*)

Set default parameters for tabular pipeline constructor.

Parameters
****kwargs** (Any) – Additional parameters.

get_cols_for_datetime(*train*)

Get datetime columns to calculate features.

Parameters
train (*SparkDataset*) – Dataset with train data.

Return type
Tuple[List[str], List[str]]

Returns
2 list of features names - base dates and common dates.

get_datetime_diffs(*train*)

Difference for all datetimes with base date.

Parameters
train (*SparkDataset*) – Dataset with train data.

Return type
Optional[*SparkBaseTransformer*]

Returns
Transformer or None if no required features.

get_datetime_seasons(*train*, *outp_role=None*)

Get season params from dates.

Parameters

- **train** (*SparkDataset*) – Dataset with train data.
- **outp_role** (Optional[ColumnRole]) – Role associated with output features.

Return type
Optional[*SparkBaseEstimator*]

Returns
Transformer or None if no required features.

get_numeric_data(*train*, *feats_to_select=None*, *prob=None*)

Select numeric features.

Parameters

- **train** (*SparkDataset*) – Dataset with train data.
- **feats_to_select** (Optional[List[str]]) – Features to handle. If None - default filter.
- **prob** (Optional[bool]) – Probability flag.

Return type
Optional[*SparkBaseTransformer*]

Returns
Transformer.

get_freq_encoding(*train*, *feats_to_select=None*)

Get frequency encoding part.

Parameters

- **train** (*SparkDataset*) – Dataset with train data.
- **feats_to_select** (Optional[List[str]]) – Features to handle. If None - default filter.

Return type
Optional[*SparkBaseEstimator*]

Returns
Transformer.

get_ordinal_encoding(*train*, *feats_to_select=None*)

Get order encoded part.

Parameters

- **train** (*SparkDataset*) – Dataset with train data.
- **feats_to_select** (Optional[List[str]]) – Features to handle. If None - default filter.

Return type
Optional[*SparkBaseEstimator*]

Returns
Transformer.

get_categorical_raw(*train*, *feats_to_select=None*)

Get label encoded categories data.

Parameters

- **train** (*SparkDataset*) – Dataset with train data.
- **feats_to_select** (Optional[List[str]]) – Features to handle. If None - default filter.

Return type
Optional[*SparkBaseEstimator*]

Returns
Transformer.

`get_target_encoder`(*train*)

Get target encoder func for dataset.

Parameters
train (*SparkDataset*) – Dataset with train data.

Return type
Optional[type]

Returns
Class

`get_binned_data`(*train*, *feats_to_select=None*)

Get encoded quantiles of numeric features.

Parameters

- **train** (*SparkDataset*) – Dataset with train data.
- **feats_to_select** (Optional[List[str]]) – features to hanlde. If None - default filter.

Return type
Optional[*SparkBaseEstimator*]

Returns
Transformer.

`get_categorical_intersections`(*train*, *feats_to_select=None*)

Get transformer that implements categorical intersections.

Parameters

- **train** (*SparkDataset*) – Dataset with train data.
- **feats_to_select** (Optional[List[str]]) – features to handle. If None - default filter.

Return type
Optional[*SparkBaseEstimator*]

Returns
Transformer.

`get_uniques_cnt`(*train*, *feats*)

Get unique values cnt.

Be aware that this function uses approx_count_distinct and thus cannot return precise results

Parameters

- **train** (*SparkDataset*) – Dataset with train data.
- **feats** (List[str]) – Features names.

Return type
Series

Returns
Series.

`get_top_categories`(*train*, *top_n=5*)

Get top categories by importance.

If feature importance is not defined, or feats has same importance - sort it by unique values counts. In second case init param `ascending_by_cardinality` defines how - asc or desc.

Parameters

- **train** (*SparkDataset*) – Dataset with train data.
- **top_n** (int) – Number of top categories.

Return type
List[str]

Returns
List.

6.1.3 SparkEmptyFeaturePipeline

class sparklightautoml.pipelines.features.base.**SparkEmptyFeaturePipeline**(***kwargs*)

Bases: *SparkFeaturesPipeline*

This class creates pipeline with SparkNoOpTransformer

create_pipeline(*train*)

Returns SparkNoOpTransformer instance

Return type
Union[*SparkBaseEstimator*, *SparkBaseTransformer*, *SparkUnionTransformer*, *SparkSequentialTransformer*]

6.1.4 SparkNoOpTransformer

class sparklightautoml.pipelines.features.base.**SparkNoOpTransformer**(*roles*)

Bases: *SparkBaseTransformer*, CommonPickleMLWritable, CommonPickleMLReadable

This transformer does nothing, it just returns the input dataframe unchanged.

6.1.5 FittedPipe

class sparklightautoml.pipelines.features.base.**FittedPipe**(*dataset*, *transformer*)

Bases: object

6.2 Feature Pipelines for Boosting Models

SparkLGBSimpleFeatures	Creates simple pipeline for tree based models.
SparkLGBAdvancedPipeline	Create advanced pipeline for trees based models.

6.2.1 SparkLGBSimpleFeatures

class sparklightautoml.pipelines.features.lgb_pipeline.**SparkLGBSimpleFeatures**

Bases: *SparkFeaturesPipeline*, *SparkTabularDataFeatures*

Creates simple pipeline for tree based models.

Simple but is ok for select features. Numeric stay as is, Datetime transforms to numeric. Categorical label encoding. Maps input to output features exactly one-to-one.

create_pipeline(*train*)

Create tree pipeline.

Parameters

train (*SparkDataset*) – Dataset with train features.

Return type

Union[*SparkUnionTransformer*, *SparkSequentialTransformer*]

Returns

Composite datetime, categorical, numeric transformer.

6.2.2 SparkLGBAdvancedPipeline

class sparklightautoml.pipelines.features.lgb_pipeline.**SparkLGBAdvancedPipeline**(*feats_imp=None*, *top_intersections=5*, *max_intersection_depth=3*, *subsample=None*, *multiclass_te_co=3*, *auto_unique_co=10*, *output_categories=False*, ***kwargs*)

Bases: *SparkFeaturesPipeline*, *SparkTabularDataFeatures*

Create advanced pipeline for trees based models.

Includes:

- Different cats and numbers handling according to role params.
- Dates handling - extracting seasons and create datediffs.
- Create categorical intersections.

__init__(*feats_imp=None*, *top_intersections=5*, *max_intersection_depth=3*, *subsample=None*, *multiclass_te_co=3*, *auto_unique_co=10*, *output_categories=False*, ***kwargs*)

Parameters

- **feats_imp** (Optional[ImportanceEstimator]) – Features importances mapping.
- **top_intersections** (int) – Max number of categories to generate intersections.
- **max_intersection_depth** (int) – Max depth of cat intersection.
- **subsample** (Union[int, float, None]) – Subsample to calc data statistics.

- **multiclass_te_co** (int) – Cutoff if use target encoding in cat handling on multiclass task if number of classes is high.
- **auto_unique_co** (int) – Switch to target encoding if high cardinality.

create_pipeline(*train*)

Create tree pipeline.

Parameters
train (*SparkDataset*) – Dataset with train features.

Return type
Union[*SparkBaseEstimator*, *SparkBaseTransformer*, *SparkUnionTransformer*, *SparkSequentialTransformer*]

Returns
Transformer.

6.3 Feature Pipelines for Linear Models

SparkLinearFeatures	Creates pipeline for linear models and nnets.

6.3.1 SparkLinearFeatures

class sparklightautoml.pipelines.features.linear_pipeline.**SparkLinearFeatures**(*feats_imp=None*, *top_intersections=5*, *max_bin_count=10*, *max_intersection_depth=3*, *subsample=None*, *sparse_ohe='auto'*, *auto_unique_co=50*, *output_categories=True*, *multiclass_te_co=3*, ***_*)

Bases: *SparkFeaturesPipeline*, *SparkTabularDataFeatures*

Creates pipeline for linear models and nnets.

Includes:

- Create categorical intersections.
- OHE or embed idx encoding for categories.
- Other cats to numbers ways if defined in role params.
- Standartization and nan handling for numbers.
- Numbers discretization if needed.
- Dates handling.
- Handling probs (output of lower level models).

__init__(*feats_imp=None*, *top_intersections=5*, *max_bin_count=10*, *max_intersection_depth=3*, *subsample=None*, *sparse_ohe='auto'*, *auto_unique_co=50*, *output_categories=True*, *multiclass_te_co=3*, ***_*)

Parameters

- **feats_imp** (Optional[ImportanceEstimator]) – Features importances mapping.
- **top_intersections** (int) – Max number of categories to generate intersections.
- **max_bin_count** (int) – Max number of bins to discretize numbers.
- **max_intersection_depth** (int) – Max depth of cat intersection.
- **subsample** (Union[int, float, None]) – Subsample to calc data statistics.
- **sparse_ohe** (Union[str, bool]) – Should we output sparse if ohe encoding was used during cat handling.
- **auto_unique_co** (int) – Switch to target encoding if high cardinality.
- **output_categories** (bool) – Output encoded categories or embed idxs.
- **multiclass_te_co** (int) – Cutoff if use target encoding in cat handling on multiclass task if number of classes is high.

create_pipeline(*train*)

Create linear pipeline.

Parameters

train (*SparkDataset*) – Dataset with train features.

Return type

Union[*SparkBaseEstimator*, *SparkBaseTransformer*, *SparkUnionTransformer*, *SparkSequentialTransformer*]

Returns

Transformer.

6.4 Utility Functions

build_graph	Fill dict that represents graph of estimators and transformers

6.4.1 build_graph

sparklightautoml.pipelines.features.base.**build_graph**(*begin*)

Fill dict that represents graph of estimators and transformers

Parameters

begin (*SparkEstOrTrans*) – pipeline to extract graph of estimators and transformers

CHAPTER

SEVEN

SPARKLIGHTAUTOML.PIPELINES.ML

Pipelines that merge together single model training steps.

7.1 Base Classes

SparkMLPipeline	Spark version of `MLPipeline`.

7.1.1 SparkMLPipeline

class sparklightautoml.pipelines.ml.base.**SparkMLPipeline**(*ml_algos*, *force_calc=True*, *pre_selection=None*, *features_pipeline=None*, *post_selection=None*, *name=None*, *persist_before_ml_algo=False*, *computations_settings=None*)

Bases: `MLPipeline`, `TransformerInputOutputRoles`

Spark version of `MLPipeline`. Single ML pipeline.

Merge together stage of building ML model (every step, excluding model training, is optional):

- Pre selection: select features from input data. Performed by `SelectionPipeline`.
- Features generation: build new features from selected. Performed by *SparkFeaturesPipeline*.
- Post selection: One more selection step - from created features. Performed by `SelectionPipeline`.
- Hyperparams optimization for one or multiple ML models. Performed by `ParamsTuner`.
- Train one or multiple ML models: Performed by *SparkTabularMLAlgo*. This step is the only required for at least 1 model.

fit_predict(*train_valid*)

Fit on train/valid iterator and transform on validation part.

Parameters
train_valid (*SparkBaseTrainValidIterator*) – Dataset iterator.

Return type
SparkDataset

Returns
Dataset with predictions of all models.

predict(*dataset*)

Predict on new dataset.

Parameters
dataset (*SparkDataset*) – Dataset used for prediction.

Return type
SparkDataset

Returns
Dataset with predictions of all trained models.

7.2 Pipeline for Nested Cross-Validation

SparkNestedTabularMLPipeline	Same as NestedTabularMLPipeline of LAMA, but redefines a couple of methods via SparkMLPipelineMixin

7.2.1 SparkNestedTabularMLPipeline

class sparklightautoml.pipelines.ml.nested_ml_pipe.**SparkNestedTabularMLPipeline**(*ml_algos*, *force_calc=True*, *pre_selection=None*, *features_pipeline=None*, *post_selection=None*, *cv=1*, *n_folds=None*, *inner_tune=False*, *refit_tuner=False*, *computations_settings=None*)

Bases: *SparkMLPipeline*, NestedTabularMLPipeline

Same as NestedTabularMLPipeline of LAMA, but redefines a couple of methods via SparkMLPipelineMixin

CHAPTER

EIGHT

SPARKLIGHTAUTOML.READER

Utils for reading, training and analysing data.

8.1 Readers

SparkToSparkReader	Reader to convert `DataFrame` to AutoML's `PandasDataset`.
SparkToSparkReaderTransformer	Transformer of SparkToSparkReader.
SparkReaderHelper	Helper class that provide some methods for *SparkToSparkReader* and *SparkToSparkReaderTransformer*.

8.1.1 SparkToSparkReader

class sparklightautoml.reader.base.**SparkToSparkReader**(*task*, *samples=100000*, *max_nan_rate=0.999*, *max_constant_rate=0.999*, *cv=5*, *random_state=42*, *roles_params=None*, *n_jobs=4*, *advanced_roles=True*, *numeric_unique_rate=0.999*, *max_to_3rd_rate=1.1*, *binning_enc_rate=2*, *raw_decr_rate=1.1*, *max_score_rate=0.2*, *abs_score_val=0.04*, *drop_score_co=0.01*, ***kwargs*)

Bases: `Reader`, *SparkReaderHelper*

Reader to convert `DataFrame` to AutoML's `PandasDataset`. Stages:

- Drop obviously useless features.
- Convert roles dict from user format to automl format.
- Simple role guess for features without input role.
- Create cv folds.
- Create initial PandasDataset.
- Optional: advanced guessing of role and handling types.

__init__(*task*, *samples=100000*, *max_nan_rate=0.999*, *max_constant_rate=0.999*, *cv=5*, *random_state=42*, *roles_params=None*, *n_jobs=4*, *advanced_roles=True*, *numeric_unique_rate=0.999*, *max_to_3rd_rate=1.1*, *binning_enc_rate=2*, *raw_decr_rate=1.1*, *max_score_rate=0.2*, *abs_score_val=0.04*, *drop_score_co=0.01*, ***kwargs*)

Parameters

- **task** (Task) – Task object.
- **samples** (Optional[int]) – Number of elements used when checking role type.
- **max_nan_rate** (float) – Maximum nan-rate.
- **max_constant_rate** (float) – Maximum constant rate.
- **cv** (int) – CV Folds.
- **random_state** (int) – Random seed.
- **roles_params** (Optional[dict]) – dict of params of features roles. Ex. {'numeric': {'dtype': np.float32}, 'datetime': {'date_format': '%Y-%m-%d'}} It's optional and commonly comes from config
- **n_jobs** (int) – Int number of processes.
- **advanced_roles** (bool) – Param of roles guess (experimental, do not change).
- **numeric_unqiue_rate** – Param of roles guess (experimental, do not change).
- **max_to_3rd_rate** (float) – Param of roles guess (experimental, do not change).
- **binning_enc_rate** (float) – Param of roles guess (experimental, do not change).
- **raw_decr_rate** (float) – Param of roles guess (experimental, do not change).
- **max_score_rate** (float) – Param of roles guess (experimental, do not change).
- **abs_score_val** (float) – Param of roles guess (experimental, do not change).
- **drop_score_co** (float) – Param of roles guess (experimental, do not change).
- ****kwargs** (Any) – For now not used.

read(*data*, *features_names=None*, *add_array_attrs=False*)

Read dataset with fitted metadata.

Parameters

- **data** (DataFrame) – Data.
- **features_names** (Optional[Any]) – Not used.
- **add_array_attrs** (bool) – Additional attributes, like target/group/weights/folds.

Return type
SparkDataset

Returns
Dataset with new columns.

fit_read(*train_data*, *features_names=None*, *roles=None*, *persistence_manager=None*, ***kwargs*)

Get dataset with initial feature selection.

Parameters

- **train_data** (DataFrame) – Input data.
- **features_names** (Optional[Any]) – Ignored. Just to keep signature.

- **roles** (Optional[Dict[Union[str, TypeVar(RoleType, bound= ColumnRole), None], Sequence[str]]]) – Dict of features roles in format {RoleX: ['feat0', 'feat1', ...], RoleY: 'TARGET', ...}.
- ****kwargs** (Any) – Can be used for target/group/weights.

Return type
SparkDataset

Returns
Dataset with selected features.

advanced_roles_guess(*dataset*, *manual_roles=None*)

Advanced roles guess over user's definition and reader's simple guessing.

Strategy - compute feature's NormalizedGini for different encoding ways and calc stats over results. Role is inferred by comparing performance stats with manual rules. Rule params are params of roles guess in init. Defaults are ok in general case.

Parameters

- **dataset** (*SparkDataset*) – Input PandasDataset.
- **manual_roles** (Optional[Dict[str, TypeVar(RoleType, bound= ColumnRole)]]) – Dict of user defined roles.

Return type
Dict[str, TypeVar(RoleType, bound= ColumnRole)]

Returns
Dict.

8.1.2 SparkToSparkReaderTransformer

class sparklightautoml.reader.base.**SparkToSparkReaderTransformer**(*task_name*, *class_mapping*, *used_array_attrs*, *roles*, *add_array_attrs=False*)

Bases: Transformer, *SparkReaderHelper*, CommonPickleMLWritable, CommonPickleMLReadable

Transformer of SparkToSparkReader. Allows to reuse SparkToSparkReader pipeline as a spark transformer.

8.1.3 SparkReaderHelper

class sparklightautoml.reader.base.**SparkReaderHelper**

Bases: object

Helper class that provide some methods for *SparkToSparkReader* and *SparkToSparkReaderTransformer*.

8.2 Utility functions for advanced roles guessing

get_category_roles_stat	Search for optimal processing of categorical values.
get_gini_func	Returns generator that take iterator by pandas dataframes and yield dataframes with calculated ginis.
get_null_scores	Get null scores.
get_numeric_roles_stat	Calculate statistics about different encodings performances.
get_score_from_pipe	Get normalized gini index from pipeline.

8.2.1 get_category_roles_stat

sparklightautoml.reader.guess_roles.**get_category_roles_stat**(*train*, *subsample=100000*, *random_state=42*)

Search for optimal processing of categorical values.

Categorical means defined by user or object types.

Parameters

- **train** (*SparkDataset*) – Dataset.
- **subsample** (Union[float, int, None]) – size of subsample.
- **random_state** (int) – seed of random numbers generator.

Returns
result.

8.2.2 get_gini_func

sparklightautoml.reader.guess_roles.**get_gini_func**(*target_col*)

Returns generator that take iterator by pandas dataframes and yield dataframes with calculated ginis.

Parameters
target_col (*str*) – target column to calc ginis

8.2.3 get_null_scores

sparklightautoml.reader.guess_roles.**get_null_scores**(*train*, *feats=None*, *subsample=100000*, *random_state=42*)

Get null scores.

Parameters

- **train** (*SparkDataset*) – Dataset
- **feats** (Optional[List[str]]) – list of features.
- **subsample** (Union[float, int, None]) – size of subsample.
- **random_state** (int) – seed of random numbers generator.

Return type
Series

Returns
Series.

8.2.4 get_numeric_roles_stat

sparklightautoml.reader.guess_roles.**get_numeric_roles_stat**(*train*, *subsample=100000*, *random_state=42*, *manual_roles=None*)

Calculate statistics about different encodings performances.

We need it to calculate rules about advanced roles guessing. Only for numeric data.

Parameters

- **train** (*SparkDataset*) – Dataset.
- **subsample** (Union[float, int, None]) – size of subsample.
- **random_state** (int) – int.
- **manual_roles** (Optional[Dict[str, ColumnRole]]) – Dict.

Return type
DataFrame

Returns
DataFrame.

8.2.5 get_score_from_pipe

sparklightautoml.reader.guess_roles.**get_score_from_pipe**(*train*, *pipe=None*)

Get normalized gini index from pipeline.

Parameters

- **train** (*SparkDataset*) – Spark dataset.
- **pipe** (Optional[Pipeline]) – Spark ML Estimator to obtain scores by means of.

Return type
ndarray

Returns
np.ndarray of scores.

CHAPTER

NINE

SPARKLIGHTAUTOML.REPORT

Report generators and templates.

CHAPTER

TEN

SPARKLIGHTAUTOML.TASKS

10.1 Task Class

SparkTask	Specify task (binary classification, multiclass classification, regression), metrics, losses.
SparkMetric	Spark version of metric function that implements function assessing prediction error.

10.1.1 SparkTask

class sparklightautoml.tasks.base.**SparkTask**(*name*, *loss=None*, *metric=None*, *greater_is_better=None*)

Bases: `Task`

Specify task (binary classification, multiclass classification, regression), metrics, losses.

get_dataset_metric()

Obtains a function to calculate the metric on a dataset.

Return type
`LAMLMetric`

10.1.2 SparkMetric

class sparklightautoml.tasks.base.**SparkMetric**(*name*, *metric_name*, *target_col=None*, *prediction_col=None*, *greater_is_better=True*)

Bases: `LAMLMetric`

Spark version of metric function that implements function assessing prediction error.

__init__(*name*, *metric_name*, *target_col=None*, *prediction_col=None*, *greater_is_better=True*)

Parameters

- **name** (`str`) – Name of metric.
- **target_col** (`Optional[str]`) – Name of column that stores targets
- **prediction_col** (`Optional[str]`) – Name of column that stores predictions
- **greater_is_better** (`bool`) – Whether or not higher metric value is better.

CHAPTER

ELEVEN

SPARKLIGHTAUTOML.TRANSFORMERS

Basic feature generation steps and helper utils.

11.1 Base Classes

SparkBaseEstimator	Base class for estimators from *sparklightautoml.transformers*.
SparkBaseTransformer	Base class for transformers from *sparklightautoml.transformers*.
SparkChangeRolesTransformer	Transformer that change roles for input columns.
SparkSequentialTransformer	Entity that represents sequential of transformers in preprocess pipeline.
SparkUnionTransformer	Entity that represents parallel layers (transformers) in preprocess pipeline.
SparkColumnsAndRoles	Helper and base class for *SparkBaseTransformer* and *SparkBaseEstimator*.
HasInputRoles	Mixin for param inputCols: input column names.
HasOutputRoles	Mixin for param inputCols: input column names.
DropColumnsTransformer	Transformer that drops columns from input dataframe.
PredictionColsTransformer	Converts prediction columns values from ONNX model format to LGBMCBooster format
ProbabilityColsTransformer	Converts probability columns values from ONNX model format to LGBMCBooster format

11.1.1 SparkBaseEstimator

class sparklightautoml.transformers.base.**SparkBaseEstimator**(*input_cols=None*, *input_roles=None*, *do_replace_columns=False*, *output_role=None*)

Bases: Estimator, *SparkColumnsAndRoles*, ABC

Base class for estimators from *sparklightautoml.transformers*.

11.1.2 SparkBaseTransformer

class sparklightautoml.transformers.base.**SparkBaseTransformer**(*input_cols*, *output_cols*, *input_roles*, *output_roles*, *do_replace_columns=False*)

Bases: Transformer, *SparkColumnsAndRoles*, ABC

Base class for transformers from *sparklightautoml.transformers*.

11.1.3 SparkChangeRolesTransformer

class sparklightautoml.transformers.base.**SparkChangeRolesTransformer**(*input_cols*, *input_roles*, *role=None*, *roles=None*)

Bases: *SparkBaseTransformer*, CommonPickleMLWritable, CommonPickleMLReadable

Transformer that change roles for input columns. Does not transform the input dataframe.

Note: this trasnformer cannot be applied directly to input columns of a feature pipeline

11.1.4 SparkSequentialTransformer

class sparklightautoml.transformers.base.**SparkSequentialTransformer**(*transformer_list*)

Bases: object

Entity that represents sequential of transformers in preprocess pipeline.

11.1.5 SparkUnionTransformer

class sparklightautoml.transformers.base.**SparkUnionTransformer**(*transformer_list*)

Bases: object

Entity that represents parallel layers (transformers) in preprocess pipeline.

get_output_cols()

Get list of output columns from all stages

Returns
output columns

Return type
List[str]

get_output_roles()

Get output roles from all stages

Returns
output roles

Return type
RolesDict

11.1.6 SparkColumnsAndRoles

class sparklightautoml.transformers.base.**SparkColumnsAndRoles**

Bases: HasInputCols, HasOutputCols, *HasInputRoles*, *HasOutputRoles*

Helper and base class for *SparkBaseTransformer* and *SparkBaseEstimator*.

11.1.7 HasInputRoles

class sparklightautoml.transformers.base.**HasInputRoles**

Bases: Params

Mixin for param inputCols: input column names.

get_input_roles()

Gets the value of inputCols or its default value.

11.1.8 HasOutputRoles

class sparklightautoml.transformers.base.**HasOutputRoles**

Bases: Params

Mixin for param inputCols: input column names.

get_output_roles()

Gets the value of inputCols or its default value.

11.1.9 DropColumnsTransformer

class sparklightautoml.transformers.base.**DropColumnsTransformer**(*remove_cols=None*, *optional_remove_cols=None*)

Bases: Transformer, DefaultParamsWritable, DefaultParamsReadable

Transformer that drops columns from input dataframe.

11.1.10 PredictionColsTransformer

class sparklightautoml.transformers.base.**PredictionColsTransformer**(*prediction_cols=None*)

Bases: Transformer, DefaultParamsWritable, DefaultParamsReadable

Converts prediction columns values from ONNX model format to LGBMCBooster format

11.1.11 ProbabilityColsTransformer

class sparklightautoml.transformers.base.**ProbabilityColsTransformer**(*probability_cols=None*, *num_classes=0*)

Bases: Transformer, DefaultParamsWritable, DefaultParamsReadable

Converts probability columns values from ONNX model format to LGBMCBooster format

11.2 Numeric

SparkFillnaMedianEstimator	Fillna with median.
SparkNaNFlagsEstimator	Estimator that calculate nan rate for input columns and build *SparkNaNFlagsTransformer*.
SparkQuantileBinningEstimator	Discretization of numeric features by quantiles.
SparkStandardScalerEstimator	Classic StandardScaler.
SparkFillInfTransformer	Transformer that replace inf values to np.nan values in input columns.
SparkFillnaMedianTransformer	Fillna with median.
SparkLogOddsTransformer	Convert probs to logodds.
SparkNaNFlagsTransformer	Adds columns with nan flags (0 or 1) for input columns.
SparkQuantileBinningTransformer	Adds column with quantile bin number of input columns.
SparkStandardScalerTransformer	Classic StandardScaler.

11.2.1 SparkFillnaMedianEstimator

class sparklightautoml.transformers.numeric.**SparkFillnaMedianEstimator**(*input_cols*, *input_roles*, *do_replace_columns=False*, *subsample=1000000*, *seed=42*)

Bases: *SparkBaseEstimator*

Fillna with median.

11.2.2 SparkNaNFlagsEstimator

class sparklightautoml.transformers.numeric.**SparkNaNFlagsEstimator**(*input_cols*, *input_roles*, *do_replace_columns=False*, *nan_rate=0.005*)

Bases: *SparkBaseEstimator*

Estimator that calculate nan rate for input columns and build *SparkNaNFlagsTransformer*.

__init__(*input_cols*, *input_roles*, *do_replace_columns=False*, *nan_rate=0.005*)

Parameters

nan_rate (float) – Nan rate cutoff.

11.2.3 SparkQuantileBinningEstimator

class sparklightautoml.transformers.numeric.**SparkQuantileBinningEstimator**(*input_cols*, *input_roles*, *do_replace_columns=False*, *nbins=10*)

Bases: *SparkBaseEstimator*

Discretization of numeric features by quantiles.

11.2.4 SparkStandardScalerEstimator

class sparklightautoml.transformers.numeric.**SparkStandardScalerEstimator**(*input_cols*, *input_roles*, *do_replace_columns=False*)

Bases: *SparkBaseEstimator*

Classic StandardScaler.

11.2.5 SparkFillInfTransformer

class sparklightautoml.transformers.numeric.**SparkFillInfTransformer**(*input_cols*, *input_roles*, *do_replace_columns=False*)

Bases: *SparkBaseTransformer*, CommonPickleMLWritable, CommonPickleMLReadable

Transformer that replace inf values to np.nan values in input columns.

11.2.6 SparkFillnaMedianTransformer

class sparklightautoml.transformers.numeric.**SparkFillnaMedianTransformer**(*input_cols*, *output_cols*, *input_roles*, *output_roles*, *meds*, *do_replace_columns=False*)

Bases: *SparkBaseTransformer*, CommonPickleMLWritable, CommonPickleMLReadable

Fillna with median.

11.2.7 SparkLogOddsTransformer

class sparklightautoml.transformers.numeric.**SparkLogOddsTransformer**(*input_cols*, *input_roles*, *do_replace_columns=False*)

Bases: *SparkBaseTransformer*, CommonPickleMLWritable, CommonPickleMLReadable

Convert probs to logodds.

11.2.8 SparkNaNFlagsTransformer

class sparklightautoml.transformers.numeric.**SparkNaNFlagsTransformer**(*input_cols*, *output_cols*, *input_roles*, *output_roles*, *do_replace_columns=False*)

Bases: *SparkBaseTransformer*, CommonPickleMLWritable, CommonPickleMLReadable

Adds columns with nan flags (0 or 1) for input columns.

11.2.9 SparkQuantileBinningTransformer

class sparklightautoml.transformers.numeric.**SparkQuantileBinningTransformer**(*bins*, *bucketizer*, *input_cols*, *output_cols*, *input_roles*, *output_roles*, *do_replace_columns=False*)

Bases: *SparkBaseTransformer*, CommonPickleMLWritable, CommonPickleMLReadable

Adds column with quantile bin number of input columns.

Quantile bin number of column value calculated by *QuantileDiscretizer* in SparkQuantileBinningEstimator.

11.2.10 SparkStandardScalerTransformer

class sparklightautoml.transformers.numeric.**SparkStandardScalerTransformer**(*input_cols*, *output_cols*, *input_roles*, *output_roles*, *means_and_stds*, *do_replace_columns=False*)

Bases: *SparkBaseTransformer*, CommonPickleMLWritable, CommonPickleMLReadable

Classic StandardScaler.

11.3 Categorical

SparkLabelEncoderEstimator	Spark label encoder estimator.
SparkOrdinalEncoderEstimator	Spark ordinal encoder estimator.
SparkFreqEncoderEstimator	Calculates frequency in train data and produces *SparkFreqEncoderTransformer* instance.
SparkCatIntersectionsEstimator	Combines categorical features and fits *SparkLabelEncoderEstimator*.
SparkTargetEncoderEstimator	Spark target encoder estimator.
SparkMulticlassTargetEncoderEstimator	Spark multiclass target encoder estimator.
SparkOHEEncoderEstimator	Simple OneHotEncoder over label encoded categories.
SparkLabelEncoderTransformer	Simple Spark version of *LabelEncoder*.
SparkOrdinalEncoderTransformer	Spark version of `OrdinalEncoder`.
SparkFreqEncoderTransformer	Labels are encoded with frequency in train data.
SparkCatIntersectionsTransformer	Combines category columns and encode with label encoder.
SparkMultiTargetEncoderTransformer	Spark multiclass target encoder transformer.
SparkCatIntersectionsHelper	Helper class for *SparkCatIntersectionsEstimator* and *SparkCatIntersectionsTransformer*.

11.3.1 SparkLabelEncoderEstimator

class sparklightautoml.transformers.categorical.**SparkLabelEncoderEstimator**(*input_cols=None*, *input_roles=None*, *subs=None*, *random_state=42*, *do_replace_columns=False*, *output_role=None*)

Bases: *SparkBaseEstimator*, `TypesHelper`

Spark label encoder estimator. Returns *SparkLabelEncoderTransformer*.

11.3.2 SparkOrdinalEncoderEstimator

class sparklightautoml.transformers.categorical.**SparkOrdinalEncoderEstimator**(*input_cols=None*, *input_roles=None*, *subs=None*, *random_state=42*)

Bases: *SparkLabelEncoderEstimator*

Spark ordinal encoder estimator. Returns *SparkOrdinalEncoderTransformer*.

11.3.3 SparkFreqEncoderEstimator

class sparklightautoml.transformers.categorical.**SparkFreqEncoderEstimator**(*input_cols*, *input_roles*, *do_replace_columns=False*)

Bases: *SparkLabelEncoderEstimator*

Calculates frequency in train data and produces *SparkFreqEncoderTransformer* instance.

11.3.4 SparkCatIntersectionsEstimator

class sparklightautoml.transformers.categorical.**SparkCatIntersectionsEstimator**(*input_cols*, *input_roles*, *intersections=None*, *max_depth=2*, *do_replace_columns=False*)

Bases: *SparkCatIntersectionsHelper*, *SparkLabelEncoderEstimator*

Combines categorical features and fits *SparkLabelEncoderEstimator*. Returns *SparkCatIntersectionsTransformer*.

11.3.5 SparkTargetEncoderEstimator

class sparklightautoml.transformers.categorical.**SparkTargetEncoderEstimator**(*input_cols*, *input_roles*, *alphas=(0.5, 1.0, 2.0, 5.0, 10.0, 50.0, 250.0, 1000.0)*, *task_name=None*, *folds_column=None*, *target_column=None*, *do_replace_columns=False*)

Bases: *SparkBaseEstimator*

Spark target encoder estimator. Returns `SparkTargetEncoderTransformer`.

11.3.6 SparkMulticlassTargetEncoderEstimator

class sparklightautoml.transformers.categorical.**SparkMulticlassTargetEncoderEstimator**(*input_cols*, *input_roles*, *alphas=(0.5, 1.0, 2.0, 5.0, 10.0, 50.0, 250.0, 1000.0)*, *task_name=None*, *folds_column=None*, *target_column=None*, *do_replace_columns=*

Bases: *SparkBaseEstimator*

Spark multiclass target encoder estimator. Returns *SparkMultiTargetEncoderTransformer*.

11.3.7 SparkOHEEncoderEstimator

class sparklightautoml.transformers.categorical.**SparkOHEEncoderEstimator**(*input_cols*, *input_roles*, *do_replace_columns=False*, *make_sparse=None*, *total_feats_cnt=None*, *dtype=numpy.float32*)

Bases: *SparkBaseEstimator*

Simple OneHotEncoder over label encoded categories.

property features

Features list.

__init__(*input_cols*, *input_roles*, *do_replace_columns=False*, *make_sparse=None*, *total_feats_cnt=None*, *dtype=numpy.float32*)

Parameters

- **make_sparse** (Optional[bool]) – Create sparse matrix.
- **total_feats_cnt** (Optional[int]) – Initial features number.
- **dtype** (type) – Dtype of new features.

11.3.8 SparkLabelEncoderTransformer

class sparklightautoml.transformers.categorical.**SparkLabelEncoderTransformer**(*input_cols*, *output_cols*, *input_roles*, *output_roles*, *do_replace_columns*, *indexer_model*)

Bases: *SparkBaseTransformer*, TypesHelper, SparkLabelEncoderTransformerMLWritable, SparkLabelEncoderTransformerMLReadable

Simple Spark version of *LabelEncoder*.

Labels are integers from 1 to n.

11.3.9 SparkOrdinalEncoderTransformer

class sparklightautoml.transformers.categorical.**SparkOrdinalEncoderTransformer**(*input_cols*, *output_cols*, *input_roles*, *output_roles*, *do_replace_columns*, *indexer_model*)

Bases: *SparkLabelEncoderTransformer*

Spark version of `OrdinalEncoder`.

Encoding ordinal categories into numbers. Number type categories passed as is, object type sorted in ascending lexicographical order.

11.3.10 SparkFreqEncoderTransformer

class sparklightautoml.transformers.categorical.**SparkFreqEncoderTransformer**(*input_cols*, *output_cols*, *input_roles*, *output_roles*, *do_replace_columns*, *indexer_model*)

Bases: *SparkLabelEncoderTransformer*

Labels are encoded with frequency in train data.

Labels are integers from 1 to n.

11.3.11 SparkCatIntersectionsTransformer

class sparklightautoml.transformers.categorical.**SparkCatIntersectionsTransformer**(*input_cols*, *output_cols*, *input_roles*, *output_roles*, *do_replace_columns*, *indexer_model*, *intersections=None*)

Bases: *SparkCatIntersectionsHelper*, *SparkLabelEncoderTransformer*

Combines category columns and encode with label encoder.

11.3.12 SparkMultiTargetEncoderTransformer

class sparklightautoml.transformers.categorical.**SparkMultiTargetEncoderTransformer**(*encodings*, *input_cols*, *output_cols*, *input_roles*, *output_roles*, *do_replace_columns=Fals*

Bases: *SparkBaseTransformer*, CommonPickleMLWritable, CommonPickleMLReadable

Spark multiclass target encoder transformer.

11.3.13 SparkCatIntersectionsHelper

class sparklightautoml.transformers.categorical.**SparkCatIntersectionsHelper**

Bases: object

Helper class for *SparkCatIntersectionsEstimator* and *SparkCatIntersectionsTransformer*.

11.4 Categorical (Scala)

laml_string_indexer.LAMLStringIndexer	Custom implementation of PySpark StringIndexer wrapper
laml_string_indexer.LAMLStringIndexerModel	Model fitted by StringIndexer.

11.4.1 LAMLStringIndexer

class sparklightautoml.transformers.scala_wrappers.laml_string_indexer.**LAMLStringIndexer**(*, *inputCol=None*, *outputCol=None*, *inputCols=None*, *outputCols=None*, *handleInvalid='error'*, *stringOrderType='frequencyD*
minFreqs=None, *defaultValue=0.0*, *freqLabel=False*, *nanLast=False*)

Bases: `JavaEstimator`, `_StringIndexerParams`, `JavaMLReadable`, `JavaMLWritable`

Custom implementation of PySpark StringIndexer wrapper

__init__(*self*, *, *inputCol=None*, *outputCol=None*, *inputCols=None*, *outputCols=None*, *handleInvalid="error"*, *stringOrderType="frequencyDesc"*)

setParams(*self*, *, *inputCol=None*, *outputCol=None*, *inputCols=None*, *outputCols=None*, *handleInvalid="error"*, *stringOrderType="frequencyDesc"*)

Sets params for this StringIndexer.

New in version 1.4.0.

setStringOrderType(*value*)

Sets the value of `stringOrderType`.

New in version 2.3.0.

setInputCol(*value*)

Sets the value of `inputCol`.

setInputCols(*value*)

Sets the value of `inputCols`.

New in version 3.0.0.

setOutputCol(*value*)

Sets the value of `outputCol`.

setOutputCols(*value*)

Sets the value of `outputCols`.

New in version 3.0.0.

setHandleInvalid(*value*)

Sets the value of `handleInvalid`.

setMinFreqs(*value*)

Sets the value of `minFreqs`.

New in version 3.2.0.

setDefaultValue(*value*)

Sets the value of `defaultValue`.

New in version 3.2.0.

setFreqLabel(*value*)

Sets the value of `freqLabel`.

New in version 3.2.0.

setNanLast(*value*)

Sets the value of `nanLast`.

New in version 3.2.0.

11.4.2 LAMLStringIndexerModel

class sparklightautoml.transformers.scala_wrappers.laml_string_indexer.**LAMLStringIndexerModel**(*java_model=*

Bases: `JavaModel`, `_StringIndexerModelParams`, `CommonJavaToPythonMLReadable`, `JavaMLWritable`

Model fitted by `StringIndexer`.

New in version 1.4.0.

setInputCol(*value*)

Sets the value of `inputCol`.

setInputCols(*value*)

Sets the value of `inputCols`.

New in version 3.0.0.

setOutputCol(*value*)

Sets the value of `outputCol`.

setOutputCols(*value*)

Sets the value of `outputCols`.

New in version 3.0.0.

setHandleInvalid(*value*)

Sets the value of `handleInvalid`.

New in version 2.4.0.

setDefaultValue(*value*)

Sets the value of `defaultValue`.

New in version 3.2.0.

setFreqLabel(*value*)

Sets the value of `freqLabel`.

New in version 3.2.0.

setNanLast(*value*)

Sets the value of `nanLast`.

New in version 3.2.0.

classmethod from_labels(*labels*, *inputCol*, *outputCol=None*, *handleInvalid=None*, *defaultValue=0.0*, *freqLabel=False*, *nanLast=False*)

Construct the model directly from an array of label strings, requires an active SparkContext.

New in version 2.4.0.

classmethod from_arrays_of_labels(*arrayOfLabels*, *inputCols*, *outputCols=None*, *handleInvalid=None*, *defaultValue=0.0*, *freqLabel=False*)

Construct the model directly from an array of array of label strings, requires an active SparkContext.

New in version 3.0.0.

property labels

Ordered list of labels, corresponding to indices to be assigned.

Deprecated since version 3.1.0: It will be removed in future versions. Use *labelsArray* method instead.

New in version 1.5.0.

property labelsArray

Array of ordered list of labels, corresponding to indices to be assigned for each input column.

New in version 3.0.2.

11.5 Datetime

SparkDateSeasonsEstimator	
SparkTimeToNumTransformer	Transforms datetime columns values to numeric values.
SparkBaseDiffTransformer	Basic conversion strategy, used in selection one-to-one transformers.
SparkDateSeasonsTransformer	Extracts unit of time from Datetime values and marks holiday dates.
SparkDatetimeHelper	Helper class for *SparkTimeToNumTransformer*, *SparkBaseDiffTransformer* and *SparkDateSeasonsTransformer*

11.5.1 SparkDateSeasonsEstimator

class sparklightautoml.transformers.datetime.**SparkDateSeasonsEstimator**(*input_cols*, *input_roles*, *output_role=None*)

Bases: *SparkBaseEstimator*

11.5.2 SparkTimeToNumTransformer

class sparklightautoml.transformers.datetime.**SparkTimeToNumTransformer**(*input_cols*, *input_roles*, *do_replace_columns=False*)

Bases: *SparkBaseTransformer*, *SparkDatetimeHelper*, CommonPickleMLWritable, CommonPickleMLReadable

Transforms datetime columns values to numeric values.

11.5.3 SparkBaseDiffTransformer

class sparklightautoml.transformers.datetime.**SparkBaseDiffTransformer**(*input_roles*, *base_names*, *diff_names*, *basic_interval='D'*, *do_replace_columns=False*)

Bases: *SparkBaseTransformer*, *SparkDatetimeHelper*, CommonPickleMLWritable, CommonPickleMLReadable

Basic conversion strategy, used in selection one-to-one transformers. Datetime converted to difference with basic_date.

11.5.4 SparkDateSeasonsTransformer

class sparklightautoml.transformers.datetime.**SparkDateSeasonsTransformer**(*input_cols*, *seasons_out_cols*, *holidays_out_cols*, *input_roles*, *output_roles*, *seasons_transformations*, *holidays_dates*)

Bases: *SparkBaseTransformer*, *SparkDatetimeHelper*, CommonPickleMLWritable, CommonPickleMLReadable

Extracts unit of time from Datetime values and marks holiday dates.

11.5.5 SparkDatetimeHelper

class sparklightautoml.transformers.datetime.**SparkDatetimeHelper**

Bases: object

Helper class for *SparkTimeToNumTransformer*, *SparkBaseDiffTransformer* and *SparkDateSeasonsTransformer*

CHAPTER

TWELVE

SPARKLIGHTAUTOML.VALIDATION

The module provide classes and functions for model validation.

12.1 Iterators

SparkBaseTrainValidIterator	Implements applying selection pipeline and feature pipeline to SparkDataset.
SparkHoldoutIterator	Simple one step iterator over one fold of SparkDataset
SparkFoldsIterator	Classic cv iterator.
SparkDummyIterator	Simple one step iterator over train part of SparkDataset

12.1.1 SparkBaseTrainValidIterator

class sparklightautoml.validation.base.**SparkBaseTrainValidIterator**(*train*)

Bases: TrainValidIterator, *Unpersistable*, ABC

Implements applying selection pipeline and feature pipeline to SparkDataset.

apply_selector(*selector*)

Select features on train data.

Check if selector is fitted. If not - fit and then perform selection. If fitted, check if it's ok to apply.

Parameters
selector (SparkSelectionPipeline) – Uses for feature selection.

Return type
SparkBaseTrainValidIterator

Returns
Dataset with selected features.

12.1.2 SparkHoldoutIterator

class sparklightautoml.validation.iterators.**SparkHoldoutIterator**(*train*)

Bases: *SparkBaseTrainValidIterator*

Simple one step iterator over one fold of SparkDataset

12.1.3 SparkFoldsIterator

class sparklightautoml.validation.iterators.**SparkFoldsIterator**(*train*, *n_folds=None*)

Bases: *SparkBaseTrainValidIterator*

Classic cv iterator.

Folds should be defined in Reader, based on cross validation method.

__init__(*train*, *n_folds=None*)

Creates iterator.

Parameters

- **train** (*SparkDataset*) – Dataset for folding.
- **n_folds** (Optional[int]) – Number of folds.

convert_to_holdout_iterator()

Convert iterator to hold-out-iterator.

Fold 0 is used for validation, everything else is used for training.

Return type
SparkHoldoutIterator

Returns
new hold-out-iterator.

12.1.4 SparkDummyIterator

class sparklightautoml.validation.iterators.**SparkDummyIterator**(*train*)

Bases: *SparkBaseTrainValidIterator*

Simple one step iterator over train part of SparkDataset

CHAPTER

THIRTEEN

RUNNING SPARK LAMA APP ON SPARK YARN WITH SPARK_SUBMIT

13.1 Prerequisites

1. Create a folder and put there the following files from the repository: * <project_root>/examples/spark/* -> examples-spark/* * <project_root>/sparklightautoml/automl/presets/tabular_config.yml -> tabular_config.yml

2. Install sparklightautoml in your python env on cluster

```
<python env on your cluster>/bin/pip install sparklightautoml
```

13.2 Launching

To launch example 'tabular-preset-automl.py' (the most comprehensive example) run the following command

```
PYSPARK_PYTHON_PATH=<python env on your cluster>
WAREHOUSE_DIR=<hdfs folder>
DRIVER_CORES=1
DRIVER_MEMORY="4g"
DRIVER_MAX_RESULT_SIZE="1g"
EXECUTOR_INSTANCES=4
EXECUTOR_CORES=4
EXECUTOR_MEMORY="10g"
CORES_MAX=$(($EXECUTOR_CORES * $EXECUTOR_INSTANCES))
# PARTITION_NUM and BUCKET_NUMS should be equal
PARTITION_NUM=$CORES_MAX
BUCKET_NUMS=$PARTITION_NUM
SCRIPT="examples-spark/tabular-preset-automl.py"

# Notes:

# "spark.kryoserializer.buffer.max=512m"
# is required when there are a lot of categorical variables with very high cardinality

# "spark.sql.autoBroadcastJoinThreshold=100MB" depends on your dataset

# if you run on jdk11
#--conf "spark.driver.extraJavaOptions=-Dio.netty.tryReflectionSetAccessible=true" \
#--conf "spark.executor.extraJavaOptions=-Dio.netty.tryReflectionSetAccessible=true"
```

(continues on next page)

(continued from previous page)

```
spark-submit \
--master yarn \
--deploy-mode cluster \
--conf "spark.yarn.appMasterEnv.SCRIPT_ENV=cluster" \
--conf "spark.yarn.appMasterEnv.PYSPARK_PYTHON=${PYSPARK_PYTHON_PATH}" \
--conf "spark.yarn.appMasterEnv.PERSISTENCE_MANAGER=CompositeBucketedPersistenceManager"␣
↪\
--conf "spark.yarn.appMasterEnv.BUCKET_NUMS=${BUCKET_NUMS}" \
--conf "spark.kryoserializer.buffer.max=512m" \
--conf "spark.driver.cores=${DRIVER_CORES}" \
--conf "spark.driver.memory=${DRIVER_MEMORY}" \
--conf "spark.driver.maxResultSize=${DRIVER_MAX_RESULT_SIZE}" \
--conf "spark.executor.instances=${EXECUTOR_INSTANCES}" \
--conf "spark.executor.cores=${EXECUTOR_CORES}" \
--conf "spark.executor.memory=${EXECUTOR_MEMORY}" \
--conf "spark.cores.max=${CORES_MAX}" \
--conf "spark.memory.fraction=0.8" \
--conf "spark.sql.shuffle.partitions=${PARTITION_NUM}" \
--conf "spark.default.parallelism=${PARTITION_NUM}" \
--conf "spark.rpc.message.maxSize=1024" \
--conf "spark.sql.autoBroadcastJoinThreshold=100MB" \
--conf "spark.sql.execution.arrow.pyspark.enabled=true" \
--conf "spark.driver.extraJavaOptions=-Dio.netty.tryReflectionSetAccessible=true" \
--conf "spark.executor.extraJavaOptions=-Dio.netty.tryReflectionSetAccessible=true" \
--conf "spark.jars.repositories=https://mmlspark.azureedge.net/maven"
--conf "spark.jars.packages=com.microsoft.azure:synapseml_2.12:0.9.5,io.github.fonhorst:
↪spark-lightautoml_2.12:0.1.1"
--conf "spark.sql.warehouse.dir=${WAREHOUSE_DIR}" \
--py-files "examples-spark/*,tabular_config.yml" \
--num-executors "${EXECUTOR_INSTANCES}" \
--jars "spark-lightautoml_2.12-0.1.jar" \
"${SCRIPT}"
```

CHAPTER

FOURTEEN

RUNNING SPARK LAMA APP ON SPARK YARN

Next, it will be shown how to run the `examples/spark/tabular-preset-automl.py` script for execution on local Hadoop YARN.

Local deployment of Hadoop YARN is done using the docker-hadoop project from the https://github.com/big-data-europe/docker-hadoop repository. It consists of the following services: datanode, historyserver, namenode, nodemanager, resourcemanager. The files `docker-hadoop/nodemanager/Dockerfile`, `docker-hadoop/docker-compose.yml` have been modified and a description of the new service `docker-hadoop/spark-submit` has been added. Required tools to get started to work with docker-hadoop project: Docker, Docker Compose and GNU Make.

14.1 1. First, let's go to the LightAutoML project directory

```
node1.bdcl LightAutoML]$ ls -gG

  4096 мар 29 17:13 bin
   930 дек  1 02:20 check_docs.py
    30 дек  1 02:20 .config.yml
  4096 мар 28 08:09 dev-tools
  4096 мар 29 20:47 dist
  4096 мар 28 08:09 docker
  1808 мар 28 08:09 docker-compose.yml
  1012 мар 28 08:09 Dockerfile
  4096 мар 29 17:13 docker-hadoop
  4096 ноя 29 22:58 docs
  4096 мар 28 08:09 examples
  4096 ноя 29 22:58 imgs
  4096 мар 29 17:16 jars
 11417 дек  1 02:20 LICENSE
  4096 дек  3 01:52 lightautoml
249874 мар 28 10:57 poetry.lock
  4274 мар 28 08:09 pyproject.toml
 12744 дек  1 02:20 README.md
  4096 мар 28 08:09 scala-lightautoml-transformers
  3476 мар 29 17:13 SparkReadme.md
  4096 ноя 29 22:58 tests
```

Make sure that in the `dist` directory there is a wheel assembly and in the `jars` directory there is a jar file.

If the `dist` directory does not exist, or if there are no files in it, then you need to build lama dist files.

```
./bin/slamactl.sh build-lama-dist
```

If there are no jar file(s) in `jars` directory, then you need to build lama jar file(s).

```
./bin/slamactl.sh build-jars
```

14.2 2. Distribute lama wheel to nodemanager

Copy lama wheel file from `dist/LightAutoML-0.3.0-py3-none-any.whl` to `docker-hadoop/nodemanager/LightAutoML-0.3.0-py3-none-any.whl`. We copy the lama wheel assembly to the nodemanager Docker file, because later it will be needed in the nodemanager service to execute the pipelines that we will send to spark.

```
cp dist/LightAutoML-0.3.0-py3-none-any.whl docker-hadoop/nodemanager/LightAutoML-0.3.0-
↪py3-none-any.whl
```

14.3 3. Go to `docker-hadoop` directory

```
cd docker-hadoop
```

14.4 4. Open `docker-compose.yml` file and configure services.

```
nano docker-compose.yml
```

Edit `volumes` setting to mount directory with datasets to `nodemanager` service.

```
nodemanager1:
  image: bde2020/hadoop-nodemanager:2.0.0-hadoop3.2.1-java8-py3.9
  container_name: nodemanager
  restart: always
  ports:
    - 8042:8042
  environment:
    SERVICE_PRECONDITION: "namenode:9000 namenode:9870 datanode:9864 resourcemanager:8088"
  volumes:
  #  - /opt/spark_data:/opt/spark_data
    - /mnt/ess_storage/DN_1/storage/sber_LAMA/kaggle_used_cars_dataset:/opt/spark_data
  env_file:
    - ./hadoop.env
```

14.5 5. Open `hadoop.env` file and configure hadoop settings.

Pay attention to the highlighted settings. They need to be set in accordance with the resources of your computers.

```
YARN_CONF_yarn_scheduler_capacity_root_default_maximum___allocation___mb=16384
YARN_CONF_yarn_scheduler_capacity_root_default_maximum___allocation___vcores=8
YARN_CONF_yarn_scheduler_maximum___allocation___mb=16384
YARN_CONF_yarn_scheduler_maximum___allocation___vcores=8
YARN_CONF_yarn_resourcemanager_fs_state___store_uri=/rmstate
YARN_CONF_yarn_resourcemanager_system___metrics___publisher_enabled=true
YARN_CONF_yarn_resourcemanager_hostname=resourcemanager
YARN_CONF_yarn_resourcemanager_address=resourcemanager:8032
YARN_CONF_yarn_resourcemanager_scheduler_address=resourcemanager:8030
YARN_CONF_yarn_resourcemanager_resource__tracker_address=resourcemanager:8031
YARN_CONF_yarn_timeline___service_enabled=true
YARN_CONF_yarn_timeline___service_generic___application___history_enabled=true
YARN_CONF_yarn_timeline___service_hostname=historyserver
YARN_CONF_mapreduce_map_output_compress=true
YARN_CONF_mapred_map_output_compress_codec=org.apache.hadoop.io.compress.SnappyCodec
YARN_CONF_yarn_nodemanager_resource_memory___mb=40960
YARN_CONF_yarn_nodemanager_resource_cpu___vcores=16
YARN_CONF_yarn_nodemanager_disk___health___checker_max___disk___utilization___per___di
YARN_CONF_yarn_nodemanager_remote___app___log___dir=/app-logs
YARN_CONF_yarn_nodemanager_aux___services=mapreduce_shuffle
```

14.6 6. Build image for `nodemanager` service.

The following command will build the `nodemanager` image according to `docker-hadoop/nodemanager/Dockerfile`. Python 3.9 and the installation of the lama wheel package has been added to this Dockerfile.

```
make build-nodemanager-with-python
```

14.7 7. Build image for `spark-submit` service.

The `spark-submit` container will be used to submit our applications for execution.

```
make build-image-to-spark-submit
```

14.8 8. Start Hadoop YARN services

```
docker-compose up
```

or same in detached mode:

```
docker-compose up -d
```

Check that all services have started:

```
docker-compose ps
```

```
[againetdinov@node1.bdcl docker-hadoop]$ docker-compose ps
     Name                   Command               State                        Ports
-------------------------------------------------------------------------------------------------
datanode            /entrypoint.sh /run.sh   Up (healthy)   9864/tcp
historyserver       /entrypoint.sh /run.sh   Up (healthy)   0.0.0.0:8188->8188/tcp
namenode            /entrypoint.sh /run.sh   Up (healthy)   0.0.0.0:9000->9000/tcp,
                                                            0.0.0.0:9870->9870/tcp
nodemanager         /entrypoint.sh /run.sh   Up (healthy)   0.0.0.0:8042->8042/tcp
resourcemanager     /entrypoint.sh /run.sh   Up (healthy)   0.0.0.0:8030->8030/tcp,
                                                            0.0.0.0:8031->8031/tcp,
                                                            0.0.0.0:8032->8032/tcp,
                                                            0.0.0.0:8088->8088/tcp
spark-submit        sleep infinity           Up
```

Here `datanode`, `historyserver`, `namenode`, `nodemanager`, `resourcemanager` is services of Hadoop. `namenode` and `datanode` is parts of HDFS. `resourcemanager`, `nodemanager` and `historyserver` is parts of YARN. For more information see the documentation at https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html and https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html.

`spark-submit` is service to submitting our applications to Hadoop YARN for execution (see step 9).

If one of the services did not up, then you need to look at its logs. For example `resourcemanager` logs.

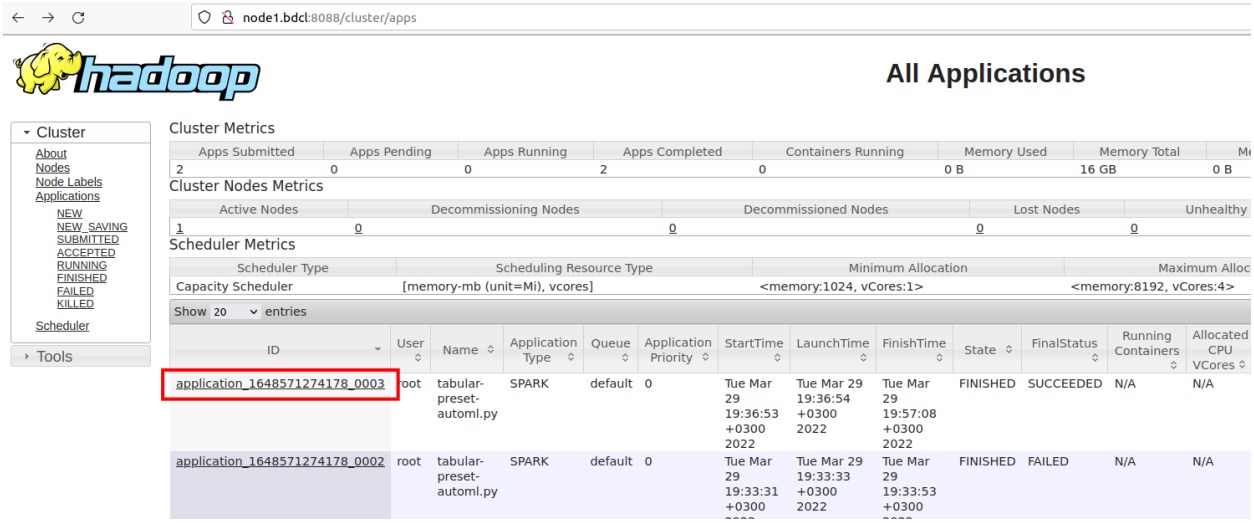
```
docker-compose logs -f resourcemanager
```

14.9 9. Send job to cluster via `spark-submit` container

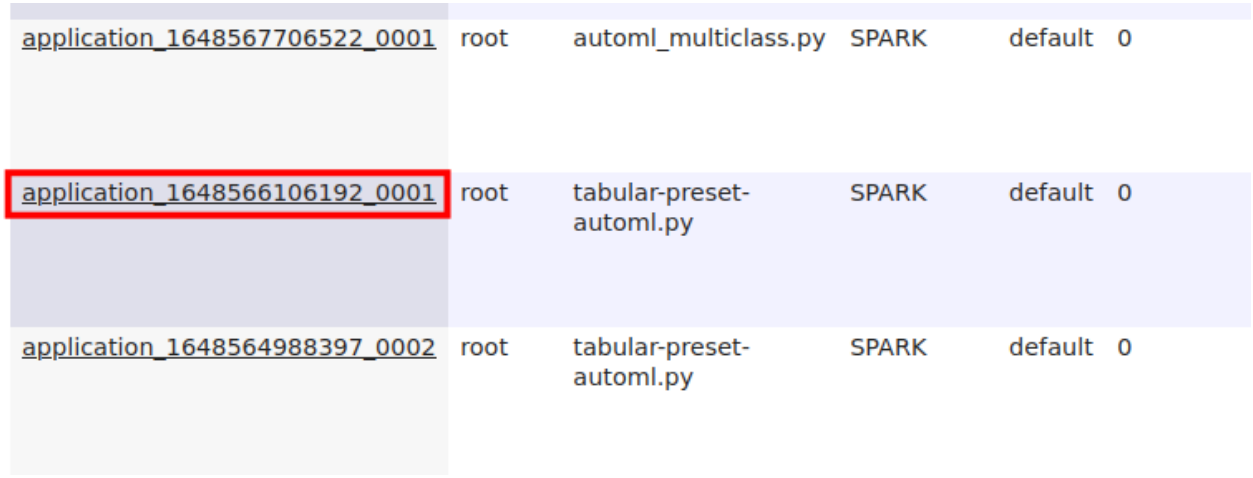
```
docker exec -ti spark-submit bash -c "./bin/slamactl.sh submit-job-yarn dist/LightAutoML-
↪0.3.0.tar.gz,examples/spark/examples_utils.py examples/spark/tabular-preset-automl.py"
```

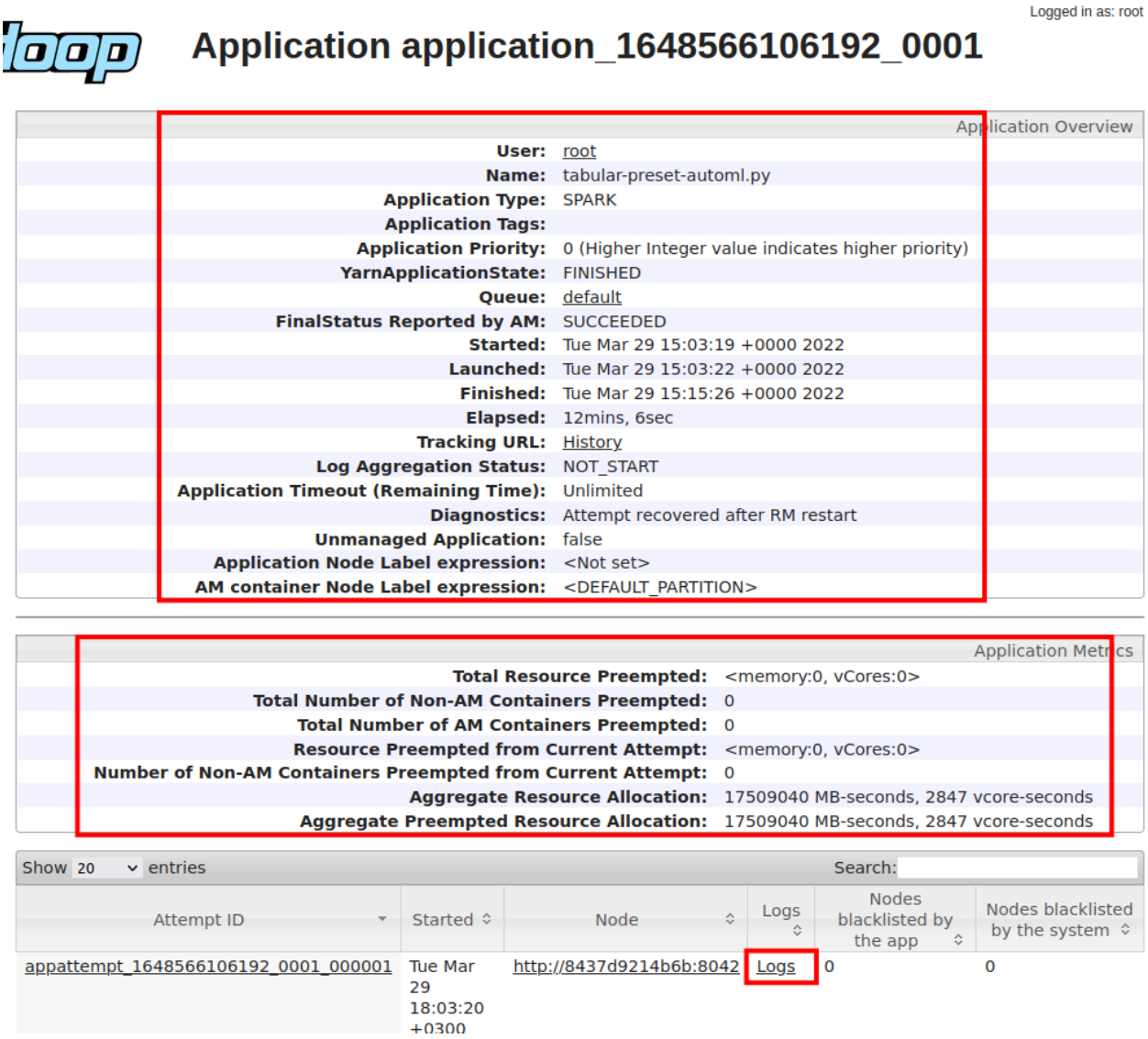
14.10 10. Monitoring application execution

To monitor application execution, you can use the hadoop web interface (http://localhost:8088), which displays the status of the application, resources and application logs.



Let's see the information about the application and its logs.





Logged in as: root

Application application_1648566106192_0001

Application Overview

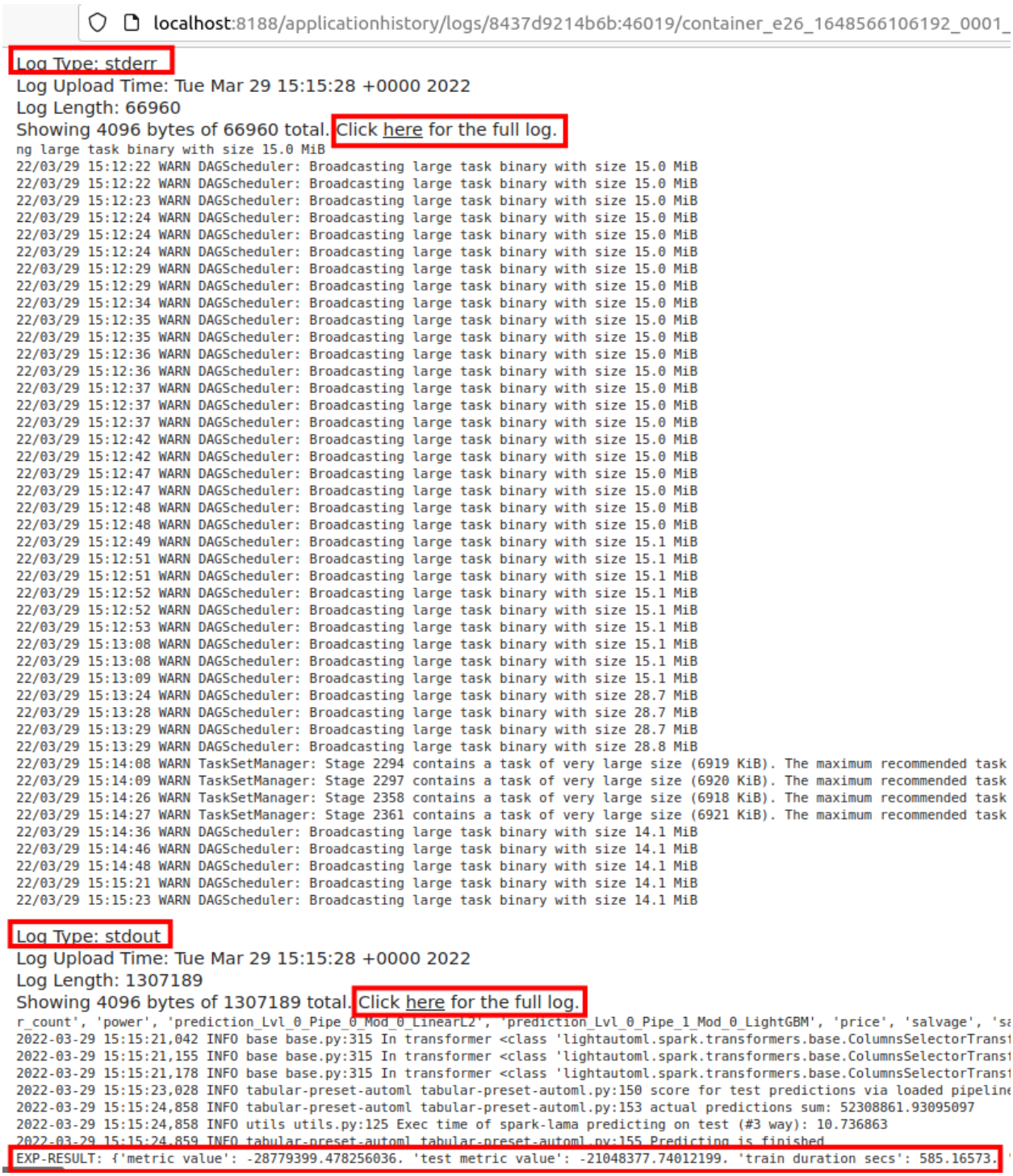
User:	root
Name:	tabular-preset-automl.py
Application Type:	SPARK
Application Tags:	
Application Priority:	0 (Higher Integer value indicates higher priority)
YarnApplicationState:	FINISHED
Queue:	default
FinalStatus Reported by AM:	SUCCEEDED
Started:	Tue Mar 29 15:03:19 +0000 2022
Launched:	Tue Mar 29 15:03:22 +0000 2022
Finished:	Tue Mar 29 15:15:26 +0000 2022
Elapsed:	12mins, 6sec
Tracking URL:	History
Log Aggregation Status:	NOT_START
Application Timeout (Remaining Time):	Unlimited
Diagnostics:	Attempt recovered after RM restart
Unmanaged Application:	false
Application Node Label expression:	<Not set>
AM container Node Label expression:	<DEFAULT_PARTITION>

Application Metrics

Total Resource Preempted:	<memory:0, vCores:0>
Total Number of Non-AM Containers Preempted:	0
Total Number of AM Containers Preempted:	0
Resource Preempted from Current Attempt:	<memory:0, vCores:0>
Number of Non-AM Containers Preempted from Current Attempt:	0
Aggregate Resource Allocation:	17509040 MB-seconds, 2847 vcore-seconds
Aggregate Preempted Resource Allocation:	17509040 MB-seconds, 2847 vcore-seconds

Show 20 entries Search:

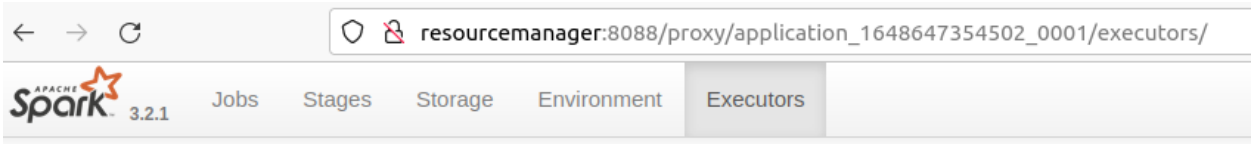
Attempt ID	Started	Node	Logs	Nodes blacklisted by the app	Nodes blacklisted by the system
appattempt_1648566106192_0001_000001	Tue Mar 29 18:03:20 +0300	http://8437d9214b6b:8042	Logs	0	0


localhost:8188/applicationhistory/logs/8437d9214b6b:46019/container_e26_1648566106192_0001_
Log Type: stderr
Log Upload Time: Tue Mar 29 15:15:28 +0000 2022
Log Length: 66960
Showing 4096 bytes of 66960 total. Click here for the full log.
ng large task binary with size 15.0 MiB
22/03/29 15:12:22 WARN DAGScheduler: Broadcasting large task binary with size 15.0 MiB
22/03/29 15:12:22 WARN DAGScheduler: Broadcasting large task binary with size 15.0 MiB
22/03/29 15:12:23 WARN DAGScheduler: Broadcasting large task binary with size 15.0 MiB
22/03/29 15:12:24 WARN DAGScheduler: Broadcasting large task binary with size 15.0 MiB
22/03/29 15:12:24 WARN DAGScheduler: Broadcasting large task binary with size 15.0 MiB
22/03/29 15:12:24 WARN DAGScheduler: Broadcasting large task binary with size 15.0 MiB
22/03/29 15:12:29 WARN DAGScheduler: Broadcasting large task binary with size 15.0 MiB
22/03/29 15:12:29 WARN DAGScheduler: Broadcasting large task binary with size 15.0 MiB
22/03/29 15:12:34 WARN DAGScheduler: Broadcasting large task binary with size 15.0 MiB
22/03/29 15:12:35 WARN DAGScheduler: Broadcasting large task binary with size 15.0 MiB
22/03/29 15:12:35 WARN DAGScheduler: Broadcasting large task binary with size 15.0 MiB
22/03/29 15:12:36 WARN DAGScheduler: Broadcasting large task binary with size 15.0 MiB
22/03/29 15:12:36 WARN DAGScheduler: Broadcasting large task binary with size 15.0 MiB
22/03/29 15:12:37 WARN DAGScheduler: Broadcasting large task binary with size 15.0 MiB
22/03/29 15:12:37 WARN DAGScheduler: Broadcasting large task binary with size 15.0 MiB
22/03/29 15:12:37 WARN DAGScheduler: Broadcasting large task binary with size 15.0 MiB
22/03/29 15:12:42 WARN DAGScheduler: Broadcasting large task binary with size 15.0 MiB
22/03/29 15:12:42 WARN DAGScheduler: Broadcasting large task binary with size 15.0 MiB
22/03/29 15:12:47 WARN DAGScheduler: Broadcasting large task binary with size 15.0 MiB
22/03/29 15:12:47 WARN DAGScheduler: Broadcasting large task binary with size 15.0 MiB
22/03/29 15:12:48 WARN DAGScheduler: Broadcasting large task binary with size 15.0 MiB
22/03/29 15:12:48 WARN DAGScheduler: Broadcasting large task binary with size 15.0 MiB
22/03/29 15:12:49 WARN DAGScheduler: Broadcasting large task binary with size 15.1 MiB
22/03/29 15:12:51 WARN DAGScheduler: Broadcasting large task binary with size 15.1 MiB
22/03/29 15:12:51 WARN DAGScheduler: Broadcasting large task binary with size 15.1 MiB
22/03/29 15:12:52 WARN DAGScheduler: Broadcasting large task binary with size 15.1 MiB
22/03/29 15:12:52 WARN DAGScheduler: Broadcasting large task binary with size 15.1 MiB
22/03/29 15:12:53 WARN DAGScheduler: Broadcasting large task binary with size 15.1 MiB
22/03/29 15:13:08 WARN DAGScheduler: Broadcasting large task binary with size 15.1 MiB
22/03/29 15:13:08 WARN DAGScheduler: Broadcasting large task binary with size 15.1 MiB
22/03/29 15:13:09 WARN DAGScheduler: Broadcasting large task binary with size 15.1 MiB
22/03/29 15:13:24 WARN DAGScheduler: Broadcasting large task binary with size 28.7 MiB
22/03/29 15:13:28 WARN DAGScheduler: Broadcasting large task binary with size 28.7 MiB
22/03/29 15:13:29 WARN DAGScheduler: Broadcasting large task binary with size 28.7 MiB
22/03/29 15:13:29 WARN DAGScheduler: Broadcasting large task binary with size 28.8 MiB
22/03/29 15:14:08 WARN TaskSetManager: Stage 2294 contains a task of very large size (6919 KiB). The maximum recommended task
22/03/29 15:14:09 WARN TaskSetManager: Stage 2297 contains a task of very large size (6920 KiB). The maximum recommended task
22/03/29 15:14:26 WARN TaskSetManager: Stage 2358 contains a task of very large size (6918 KiB). The maximum recommended task
22/03/29 15:14:27 WARN TaskSetManager: Stage 2361 contains a task of very large size (6921 KiB). The maximum recommended task
22/03/29 15:14:36 WARN DAGScheduler: Broadcasting large task binary with size 14.1 MiB
22/03/29 15:14:46 WARN DAGScheduler: Broadcasting large task binary with size 14.1 MiB
22/03/29 15:14:48 WARN DAGScheduler: Broadcasting large task binary with size 14.1 MiB
22/03/29 15:15:21 WARN DAGScheduler: Broadcasting large task binary with size 14.1 MiB
22/03/29 15:15:23 WARN DAGScheduler: Broadcasting large task binary with size 14.1 MiB
Log Type: stdout
Log Upload Time: Tue Mar 29 15:15:28 +0000 2022
Log Length: 1307189
Showing 4096 bytes of 1307189 total. Click here for the full log.
r_count', 'power', 'prediction_Lvl_0_Pipe_0_Mod_0_LinearL2', 'prediction_Lvl_0_Pipe_1_Mod_0_LightGBM', 'price', 'salvage', 'sa
2022-03-29 15:15:21,042 INFO base base.py:315 In transformer <class 'lightautoml.spark.transformers.base.ColumnsSelectorTransf
2022-03-29 15:15:21,155 INFO base base.py:315 In transformer <class 'lightautoml.spark.transformers.base.ColumnsSelectorTransf
2022-03-29 15:15:21,178 INFO base base.py:315 In transformer <class 'lightautoml.spark.transformers.base.ColumnsSelectorTransf
2022-03-29 15:15:23,028 INFO tabular-preset-automl tabular-preset-automl.py:150 score for test predictions via loaded pipeline
2022-03-29 15:15:24,858 INFO tabular-preset-automl tabular-preset-automl.py:153 actual predictions sum: 52308861.93095097
2022-03-29 15:15:24,858 INFO utils utils.py:125 Exec time of spark-lama predicting on test (#3 way): 10.736863
2022-03-29 15:15:24,859 INFO tabular-preset-automl tabular-preset-automl.py:155 Predicting is finished
EXP-RESULT: {'metric value': -28779399.478256036, 'test metric value': -21048377.74012199, 'train duration secs': 585.16573,

14.11 11. Spark WebUI

When the application is running, you can go to the hadoop web interface and get a link to the Spark WebUI.

User:	root
Name:	tabular-preset-automl.py
Application Type:	SPARK
Application Tags:	
Application Priority:	0 (Higher Integer value indicates higher priority)
YarnApplicationState:	ACCEPTED: waiting for AM container to be allocated, launched and register with RM.
Queue:	default
FinalStatus Reported by AM:	Application has not completed yet.
Started:	Wed Mar 30 14:00:18 +0000 2022
Launched:	Wed Mar 30 14:00:19 +0000 2022
Finished:	N/A
Elapsed:	10sec
Tracking URL:	ApplicationMaster
Log Aggregation Status:	NOT_START
Application Timeout (Remaining Time):	Unlimited
Diagnostics:	AM container is launched, waiting for AM container to Register with RM
Unmanaged Application:	false
Application Node Label expression:	<Not set>
AM container Node Label expression:	<DEFAULT_PARTITION>



Executors

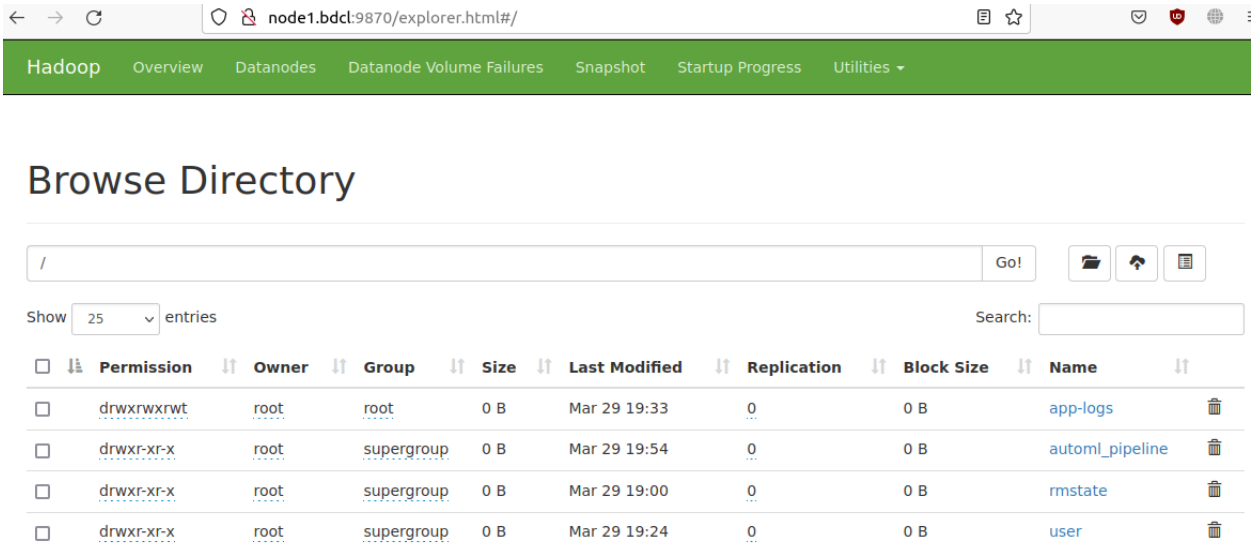
Show Additional Metrics

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks
Active(6)	0	0.0 B / 14.9 GiB	0.0 B	40	0	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0
Total(6)	0	0.0 B / 14.9 GiB	0.0 B	40	0	0

14.12 12. HDFS Web UI

HDFS Web UI is available at http://localhost:9870. Here you can browse your files in HDFS http://localhost:9870/explorer.html. HDFS stores trained pipelines and Spark application files.


node1.bdcl:9870/explorer.html#/
Hadoop
Overview
Datanodes
Datanode Volume Failures
Snapshot
Startup Progress
Utilities
Browse Directory
/
Go!
Show 25 entries
Search:
Permission
Owner
Group
Size
Last Modified
Replication
Block Size
Name
drwxrwxrwt root root 0 B Mar 29 19:33 0 0 B app-logs
drwxr-xr-x root supergroup 0 B Mar 29 19:54 0 0 B automl_pipeline
drwxr-xr-x root supergroup 0 B Mar 29 19:00 0 0 B rmstate
drwxr-xr-x root supergroup 0 B Mar 29 19:24 0 0 B user

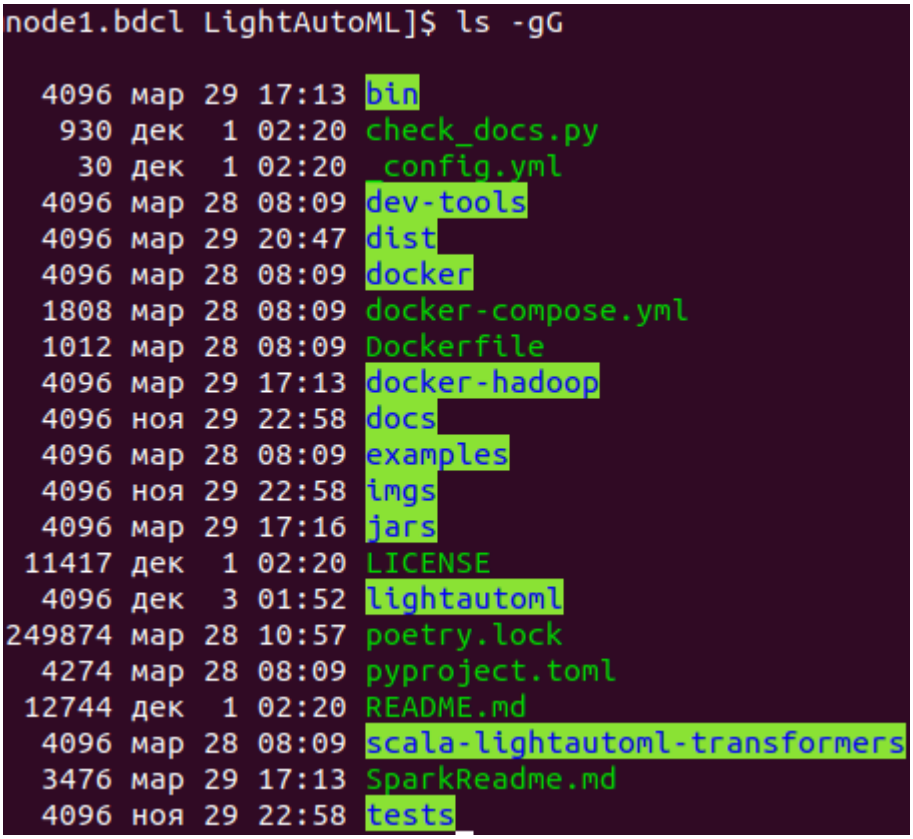
CHAPTER

FIFTEEN

RUNNING SPARK LAMA APP ON STANDALONE CLUSTER

Next, it will be shown how to run the `examples/spark/tabular-preset-automl.py` script for execution on Spark cluster.

15.1 1. First, let's go to the LightAutoML project directory



Make sure that in the `dist` directory there is a wheel assembly and in the `jars` directory there is a jar file. If the `dist` directory does not exist, or if there are no files in it, then you need to build lama dist files.

```
./bin/slamactl.sh build-lama-dist
```

If there are no jar file(s) in `jars` directory, then you need to build lama jar file(s).

```
./bin/slamactl.sh build-jars
```

15.2 2. Set Spark master URL via environment variable

```
export SPARK_MASTER_URL=spark://HOST:PORT
```

For example:

```
export SPARK_MASTER_URL=spark://node21.bdcl:7077
```

15.3 3. Set Hadoop namenode address (fs.defaultFS) via environment variable

```
export HADOOP_DEFAULT_FS=hdfs://HOST:PORT
```

For example:

```
export HADOOP_DEFAULT_FS=hdfs://node21.bdcl:9000
```

15.4 4. Submit job via `slamactl.sh` script

```
./bin/slamactl.sh submit-job-spark examples/spark/tabular-preset-automl.py
```

CHAPTER

SIXTEEN

DEPLOY ON MINIKUBE

1. On a host with Linux operating system and amd64 architecture, run the following commands:

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-
↪amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

   For other operating systems and architectures, one should follow the official guide.

2. Start minikube:

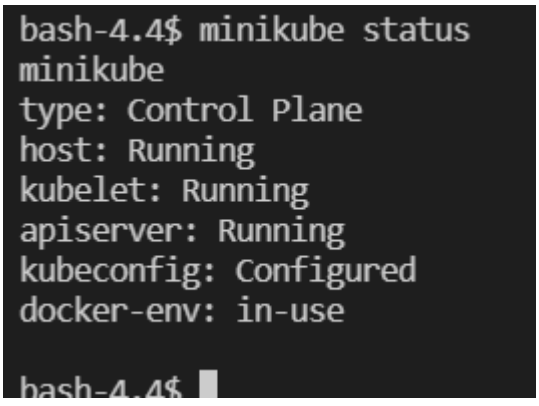
```
minikube start -cpus 12 -memory 20000 -driver=docker
```

Setting of cpus and memory is up to you.

3. Check minikube status:

```
minikube status
```

All services should be in running state:



CHAPTER

SEVENTEEN

ENVIRONMENT SETUP

1. Clone LigthAutoML repository.
2. Check Python is installed on the host.
3. Download libraries required to be installed on the host:

```
pip install pyspark
pip install poetry
```

5. Create several folders to be used for data storage and particulary for pv (PersistentVolume) and pvc (PersistentVolumeClaim): One may choose different paths. All described below is just a suggestion.
   - /opt/data-slama - dataset folder. All required datasets, one is planning to work with, should be copied in this folder.
   - /opt/result-slama - service folder for intermediate data
   - /tmp/slama - temporary files folder
6. Mount the folders mentioned earlier into minikube:

```
minikube mount <source>:<dest>
```

!! 9p

7. Create a namespace and a service account in K8s to run SLAMA application and give it ability to create executor pods. Take a look on examples in ~/LightAutoML/dev-tools/config/spark-lama-ns.yaml.

```
kubectl apply -f ./dev-tools/config/spark-lama-ns.yaml
```

Results of the command execution should look like:

```
[vagrant@control-plane LightAutoML]$ kubectl apply -f ./dev-tools/config/spark-lama-ns.yaml
namespace/spark-lama-exps unchanged
serviceaccount/spark unchanged
role.rbac.authorization.k8s.io/spark-executor-creator unchanged
rolebinding.rbac.authorization.k8s.io/spark-binding unchanged
[vagrant@control-plane LightAutoML]$
```

Instead of 'unchanged' state there may be 'created' state if nothing existed before this command was executed.

8. Create pv and pvc to be used by spark application with SLAMA. It is assumed that the folders previously created will be used for this purpose. One may take a look on the example ~/LightAutoML/dev-tools/config/spark-lama-data-pv-pvc.yaml to create pv and pvc.

```
kubectl apply -f ./dev-tools/config/spark-lama-data-pv-pvc.yaml
```

8. Setup environment variables to be used with slamactl.sh utility:

```
export KUBE_NAMESPACE=spark-lama-exps
export REPO=node2.bdcl:5000
```

9. Build required images:

```
./bin/slamactl.sh build-dist
./bin/slamactl.sh build-lama-image
```

10. One can check resulting images with the command:

    docker images

```
node2.bdcl:5000/spark-py-lama          lama-v3.2.0-nikolay   0bc6e7c604ea   15 hours ago     6.95GB
node2.bdcl:5000/spark-py               lama-v3.2.0-nikolay   24000300222d   21 hours ago     962MB
node2.bdcl:5000/spark                  lama-v3.2.0-nikolay   c97d29d5c745   21 hours ago     602MB
[vagrant@control-plane LightAutoML]$
```

11. Upload required datasets into the folder of pv spark-lama-data.

CHAPTER

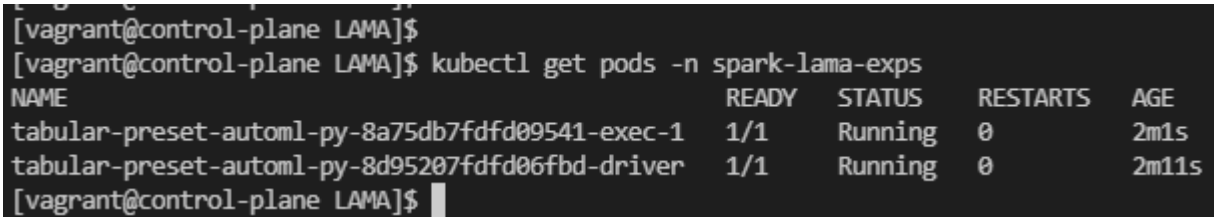
EIGHTEEN

RUN EXAMPLES IN MINIKUBE

1. Ensure that REPO and KUBE_NAMESPACE variables are set. Ensure that all required docker images and kubernetes objects have been created.
2. Go to LigthAutoML folder.
3. Run an example with slamactl.sh:

```
./bin/slamactl.sh submit-job ./examples/spark/tabular-preset-automl.py
```

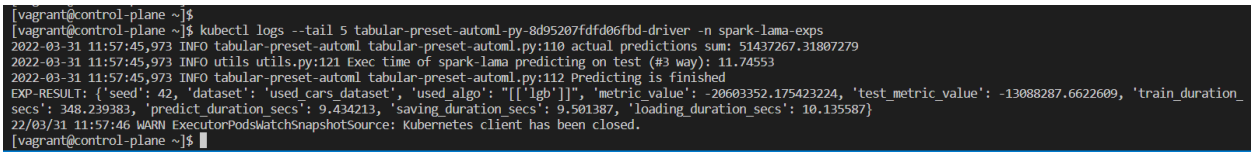
4. Check state of SLAMA application's pods with command `kubectl get pods -n spark-lama-exps`:



5. Check the result of execution when the driver pod is completed with:

```
kubectl logs --tail 5 tabular-preset-automl-py-8d95207fdfd06fbd-driver -n spark-
↪lama-exps
```

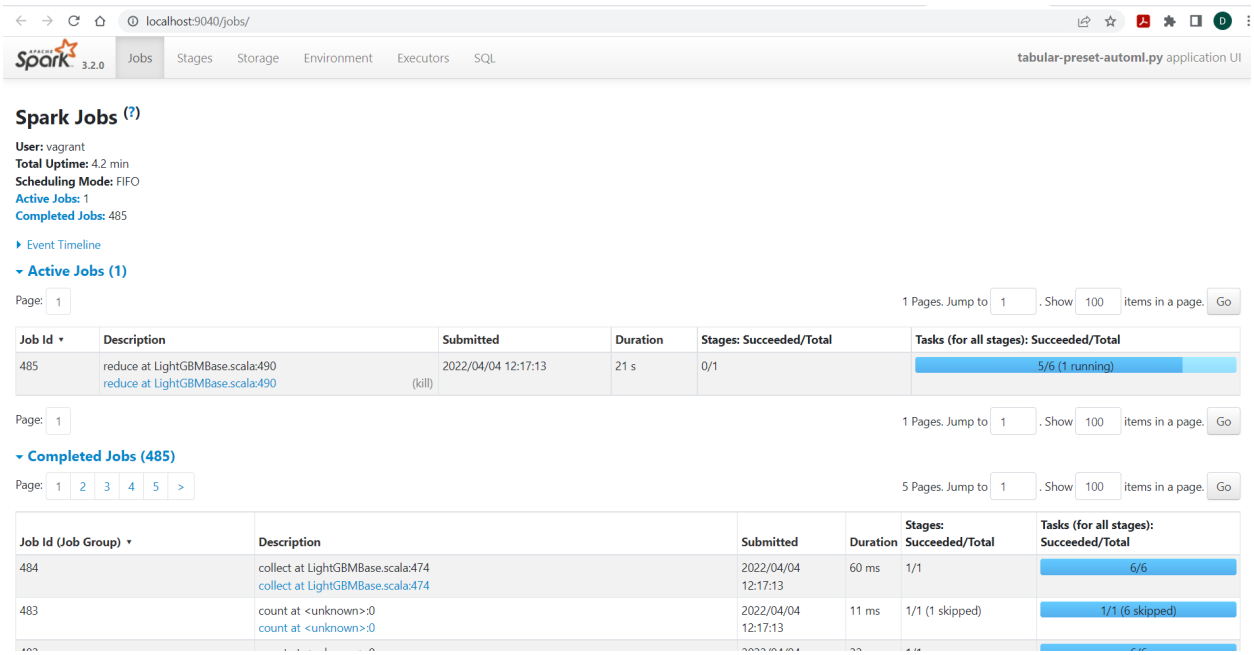
An example of the result:



6. One can open Spark Web UI of SLAMA application on localhost. That requires to execute a command for port forwarding to one of localhost ports:

```
kubectl -n spark-lama-exps port-forward svc/$(kubectl -n spark-lama-exps get svc -o␣
↪jsonpath='{.items[0].metadata.name}') 9040:4040 --address='0.0.0.0'
```

To open Spark WebUI follow the link <http://localhost:9040>



Note: SLAMA application should be in running state.

CHAPTER

NINETEEN

RUNNING SPARK LAMA APP ON KUBERNETES CLUSTER

Examples for Spark-LAMA can be found in `examples/spark/`. These examples can be run both locally and remotely on a cluster.

To run examples locally one needs just ensure that data files lay in appropriate locations. These locations typically /opt/spark_data directory. (Data for the examples can be found in examples/data)

To run examples remotely on a cluster under Kubernetes control one needs to have installed and configured **kubectl** utility.

19.1 1. Establish nfs / S3

This step is necessary to make uploading of script file (e.g. executable of Spark LAMA) into a location that is accessible from anywhere on cluster. This file will be used by spark driver which is also submitted to the cluster. Upon configuring set appropriate value for *spark.kubernetes.file.upload.path* in `./bin/slamactl.sh` or mount it to `/mnt/nfs` on the localhost.

19.2 2. Create persistent volumes (PV) and claims (PVC)

Examples required 2 PVC for their functioning (defined in slamactl.sh, spark-submit arguments):

- *spark-lama-data* - provides access for driver and executors to data
- *mnt-nfs* - provide access for driver and executors to the mentioned above upload dir

19.3 3. Define required env variables

Define required environment variables to use appropriate kubernetes namespace and remote docker repository accessible from anywhere in the cluster.

```
export KUBE_NAMESPACE=spark-lama-exps
export REPO=node2.bdcl:5000
```

19.4 4. Build spark lama dependencies and docker images.

On this step use slamactl.sh utility to build dependencies and docker images:

```
./bin/slamactl.sh build-dist
```

It will:

- compile jars containing Scala-based components

(currently only LAMLStringIndexer required for LE-family transformers)

- download Spark distro and use dockerfiles from there to build base pyspark images

(and push these images to the remote docker repo)

- compile lama wheel (including spark subpackage) and build a docker image based upon mentioned above pyspark images

(this image will be pushed to the remote repository too)

19.5 5. Run an example on the remote cluster

To do that use the following command:

```
./bin/slamactl.sh submit-job ./examples/spark/tabular-preset-automl.py
```

The command submits a driver pod (using spark-submit) to the cluster which creates executor pods.

19.6 6. Forward 4040 port to make Spark Web UI accessible.

The utility provides a command to make port forwording for the running example.

```
./bin/slamactl.sh port-forward ./examples/spark/tabular-preset-automl.py
```

The driver's 4040 port will be forwarded to http://localhost:9040.

INDEX

Symbols

__init__() (*sparklightautoml.automl.base.SparkAutoML method*), 3
__init__() (*sparklightautoml.automl.presets.base.SparkAutoMLPreset method*), 6
__init__() (*sparklightautoml.dataset.roles.NumericVectorOrArrayRole method*), 15
__init__() (*sparklightautoml.ml_algo.base.AveragingTransformer method*), 20
__init__() (*sparklightautoml.pipelines.features.base.SparkTabularDataFeatures method*), 30
__init__() (*sparklightautoml.pipelines.features.lgb_pipeline.SparkLGBAdvancedPipeline method*), 34
__init__() (*sparklightautoml.pipelines.features.linear_pipeline.SparkLinearFeatures method*), 35
__init__() (*sparklightautoml.pipelines.selection.permutation_importance_based.SparkNpPermutationImportanceEstimator method*), 27
__init__() (*sparklightautoml.reader.base.SparkToSparkReader method*), 39
__init__() (*sparklightautoml.tasks.base.SparkMetric method*), 47
__init__() (*sparklightautoml.transformers.categorical.SparkOHEEncoderEstimator method*), 57
__init__() (*sparklightautoml.transformers.numeric.SparkNaNFlagsEstimator method*), 52
__init__() (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStringIndexer method*), 60
__init__() (*sparklightautoml.validation.iterators.SparkFoldsIterator method*), 66

A

advanced_roles_guess() (*sparklightautoml.reader.base.SparkToSparkReader method*), 41
all_datasets (*sparklightautoml.dataset.base.PersistenceManager property*), 14
apply_selector() (*sparklightautoml.validation.base.SparkBaseTrainValidIterator method*), 65
AveragingTransformer (*class in sparklightautoml.ml_algo.base*), 20

B

BasePersistenceManager (*class in sparklightautoml.dataset.persistence*), 17
BucketedPersistenceManager (*class in sparklightautoml.dataset.persistence*), 17
build_graph() (*in module sparklightautoml.pipelines.features.base*), 36

C

collect_model_stats() (*sparklightautoml.automl.base.SparkAutoML method*), 5
collect_used_feats() (*sparklightautoml.automl.base.SparkAutoML method*), 5
CompositeBucketedPersistenceManager (*class in sparklightautoml.dataset.persistence*), 18
CompositePersistenceManager (*class in sparklightautoml.dataset.persistence*), 18
concatenate() (*sparklightautoml.dataset.base.SparkDataset class method*), 13
convert_to_holdout_iterator() (*sparklightautoml.validation.iterators.SparkFoldsIterator method*), 66
create_automl() (*sparklightautoml.automl.presets.base.SparkAutoMLPreset method*), 7

create_automl() (*sparklightautoml.automl.presets.tabular_presets.SparkTabularAutoML method*), 8
create_pipeline() (*sparklightautoml.pipelines.features.base.SparkEmptyFeaturePipeline method*), 33
create_pipeline() (*sparklightautoml.pipelines.features.base.SparkFeaturesPipeline method*), 29
create_pipeline() (*sparklightautoml.pipelines.features.lgb_pipeline.SparkLGBAdvancedPipeline method*), 35
create_pipeline() (*sparklightautoml.pipelines.features.lgb_pipeline.SparkLGBSimpleFeatures method*), 34
create_pipeline() (*sparklightautoml.pipelines.features.linear_pipeline.SparkLinearFeatures method*), 36

D

DropColumnsTransformer (*class in sparklightautoml.transformers.base*), 51

F

feature_name_at() (*sparklightautoml.dataset.roles.NumericVectorOrArrayRole method*), 15
features (*sparklightautoml.dataset.base.SparkDataset property*), 13
features (*sparklightautoml.ml_algo.base.SparkTabularMLAlgo property*), 19
features (*sparklightautoml.transformers.categorical.SparkOHEEncoderEstimator property*), 57
fit() (*sparklightautoml.pipelines.selection.permutation_importance_based.SparkNpPermutationImportanceEstimator method*), 27
fit_predict() (*sparklightautoml.automl.base.SparkAutoML method*), 4
fit_predict() (*sparklightautoml.automl.presets.base.SparkAutoMLPreset method*), 7
fit_predict() (*sparklightautoml.automl.presets.tabular_presets.SparkTabularAutoML method*), 8
fit_predict() (*sparklightautoml.ml_algo.boost_lgbm.SparkBoostLGBM method*), 22
fit_predict() (*sparklightautoml.pipelines.ml.base.SparkMLPipeline method*), 37
fit_predict_single_fold() (*sparklightautoml.ml_algo.base.SparkTabularMLAlgo method*), 19
fit_read() (*sparklightautoml.reader.base.SparkToSparkReader method*), 40
fit_transform() (*sparklightautoml.pipelines.features.base.SparkFeaturesPipeline method*), 29
FittedPipe (*class in sparklightautoml.pipelines.features.base*), 33
from_arrays_of_labels() (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStri class method*), 62
from_labels() (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStri class method*), 62

G

get_binned_data() (*sparklightautoml.pipelines.features.base.SparkTabularDataFeatures method*), 32
get_categorical_intersections() (*sparklightautoml.pipelines.features.base.SparkTabularDataFeatures method*), 32
get_categorical_raw() (*sparklightautoml.pipelines.features.base.SparkTabularDataFeatures method*), 31
get_category_roles_stat() (*in module sparklightautoml.reader.guess_roles*), 42
get_cols_for_datetime() (*sparklightautoml.pipelines.features.base.SparkTabularDataFeatures method*), 30
get_config() (*sparklightautoml.automl.presets.base.SparkAutoMLPreset class method*), 6
get_dataset_metric() (*sparklightautoml.tasks.base.SparkTask method*), 47
get_datetime_diffs() (*sparklightautoml.pipelines.features.base.SparkTabularDataFeatures method*), 30
get_datetime_seasons() (*sparklightautoml.pipelines.features.base.SparkTabularDataFeatures method*), 30
get_freq_encoding() (*sparklightautoml.pipelines.features.base.SparkTabularDataFeatures method*), 31
get_gini_func() (*in module sparklightautoml.reader.guess_roles*), 42
get_input_roles() (*sparklightautoml.transformers.base.HasInputRoles method*), 51
get_null_scores() (*in module sparklightautoml.reader.guess_roles*), 42
get_numeric_data() (*sparklightautoml.pipelines.features.base.SparkTabularDataFeatures*

method), 31
get_numeric_roles_stat() (*in module sparklightautoml.reader.guess_roles*), 43
get_ordinal_encoding() (*sparklightautoml.pipelines.features.base.SparkTabularDataFeatures method*), 31
get_output_cols() (*sparklightautoml.transformers.base.SparkUnionTransformer method*), 50
get_output_roles() (*sparklightautoml.transformers.base.HasOutputRoles method*), 51
get_output_roles() (*sparklightautoml.transformers.base.SparkUnionTransformer method*), 50
get_pdp_data_categorical_feature() (*sparklightautoml.automl.presets.tabular_presets.SparkTabularAutoML static method*), 10
get_pdp_data_datetime_feature() (*sparklightautoml.automl.presets.tabular_presets.SparkTabularAutoML static method*), 10
get_pdp_data_numeric_feature() (*sparklightautoml.automl.presets.tabular_presets.SparkTabularAutoML static method*), 9
get_score_from_pipe() (*in module sparklightautoml.reader.guess_roles*), 43
get_target_encoder() (*sparklightautoml.pipelines.features.base.SparkTabularDataFeatures method*), 31
get_top_categories() (*sparklightautoml.pipelines.features.base.SparkTabularDataFeatures method*), 32
get_uniques_cnt() (*sparklightautoml.pipelines.features.base.SparkTabularDataFeatures method*), 32

H

HasInputRoles (*class in sparklightautoml.transformers.base*), 51
HasOutputRoles (*class in sparklightautoml.transformers.base*), 51

I

input_roles (*sparklightautoml.automl.blend.SparkBlender property*), 11

L

labels (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStringIndexerModel property*), 62
labelsArray (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStringIndexerModel property*), 62
LAMLStringIndexer (*class in sparklightautoml.transformers.scala_wrappers.laml_string_indexer*), 60
LAMLStringIndexerModel (*class in sparklightautoml.transformers.scala_wrappers.laml_string_indexer*), 61
LightGBMModelWrapper (*class in sparklightautoml.ml_algo.boost_lgbm*), 23
LocalCheckpointPersistenceManager (*class in sparklightautoml.dataset.persistence*), 17

N

NumericVectorOrArrayRole (*class in sparklightautoml.dataset.roles*), 15

O

ONNXModelWrapper (*class in sparklightautoml.ml_algo.boost_lgbm*), 23
output_roles (*sparklightautoml.automl.blend.SparkBlender property*), 11

P

persist() (*sparklightautoml.dataset.base.SparkDataset method*), 14
PersistenceLevel (*class in sparklightautoml.dataset.base*), 14
PersistenceManager (*class in sparklightautoml.dataset.base*), 14
PlainCachePersistenceManager (*class in sparklightautoml.dataset.persistence*), 17
predict() (*sparklightautoml.automl.base.SparkAutoML method*), 4
predict() (*sparklightautoml.pipelines.ml.base.SparkMLPipeline method*), 37
predict_single_fold() (*sparklightautoml.ml_algo.linear_pyspark.SparkLinearLBFGS method*), 21
PredictionColsTransformer (*class in sparklightautoml.transformers.base*), 51
ProbabilityColsTransformer (*class in sparklightautoml.transformers.base*), 52

R

read() (*sparklightautoml.ml_algo.boost_lgbm.LightGBMModelWrapper class method*), 23
read() (*sparklightautoml.ml_algo.boost_lgbm.ONNXModelWrapper class method*), 23
read() (*sparklightautoml.reader.base.SparkToSparkReader method*), 40
roles (*sparklightautoml.dataset.base.SparkDataset property*), 14

S

score() (*sparklightautoml.automl.blend.SparkBlender method*), 12
set_data() (*sparklightautoml.dataset.base.SparkDataset method*), 14
set_verbosity_level() (*sparklightautoml.automl.presets.base.SparkAutoMLPreset static method*), 7
setDefaultValue() (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStringIndexer method*), 61
setDefaultValue() (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStringIndexerModel method*), 61
setFreqLabel() (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStringIndexer method*), 61
setFreqLabel() (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStringIndexerModel method*), 62
setHandleInvalid() (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStringIndexer method*), 61
setHandleInvalid() (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStringIndexerModel method*), 61
setInputCol() (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStringIndexer method*), 60
setInputCol() (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStringIndexerModel method*), 61
setInputCols() (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStringIndexer method*), 60
setInputCols() (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStringIndexerModel method*), 61
setMinFreqs() (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStringIndexer method*), 61
setNanLast() (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStringIndexer method*), 61
setNanLast() (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStringIndexerModel method*), 62
setOutputCol() (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStringIndexer method*), 60
setOutputCol() (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStringIndexerModel method*), 61
setOutputCols() (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStringIndexer method*), 61
setOutputCols() (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStringIndexerModel method*), 61
setParams() (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStringIndexer method*), 60
setStringOrderType() (*sparklightautoml.transformers.scala_wrappers.laml_string_indexer.LAMLStringIndexer method*), 60
SparkAutoML (*class in sparklightautoml.automl.base*), 3
SparkAutoMLPreset (*class in sparklightautoml.automl.presets.base*), 6
SparkBaseDiffTransformer (*class in sparklightautoml.transformers.datetime*), 63
SparkBaseEstimator (*class in sparklightautoml.transformers.base*), 49
SparkBaseTrainValidIterator (*class in sparklightautoml.validation.base*), 65
SparkBaseTransformer (*class in sparklightautoml.transformers.base*), 50
SparkBestModelSelector (*class in sparklightautoml.automl.blend*), 12
SparkBlender (*class in sparklightautoml.automl.blend*), 11
SparkBoostLGBM (*class in sparklightautoml.ml_algo.boost_lgbm*), 21
SparkCatIntersectionsEstimator (*class in sparklightautoml.transformers.categorical*), 56
SparkCatIntersectionsHelper (*class in sparklightautoml.transformers.categorical*), 59
SparkCatIntersectionsTransformer (*class in sparklightautoml.transformers.categorical*), 59
SparkChangeRolesTransformer (*class in sparklightautoml.transformers.base*), 50
SparkColumnsAndRoles (*class in sparklightautoml.transformers.base*), 51
SparkDataset (*class in sparklightautoml.dataset.base*), 13
SparkDateSeasonsEstimator (*class in sparklightautoml.transformers.datetime*), 63
SparkDateSeasonsTransformer (*class in sparklightautoml.transformers.datetime*), 63
SparkDatetimeHelper (*class in sparklightautoml.transformers.datetime*), 64
SparkDummyIterator (*class in sparklightautoml.validation.iterators*), 66
SparkEmptyFeaturePipeline (*class in sparklightautoml.pipelines.features.base*), 33
SparkFeaturesPipeline (*class in sparklightautoml.pipelines.features.base*), 29
SparkFillInfTransformer (*class in sparklightau-*

toml.transformers.numeric), 53
SparkFillnaMedianEstimator (*class in sparklightautoml.transformers.numeric*), 52
SparkFillnaMedianTransformer (*class in sparklightautoml.transformers.numeric*), 53
SparkFoldsIterator (*class in sparklightautoml.validation.iterators*), 66
SparkFreqEncoderEstimator (*class in sparklightautoml.transformers.categorical*), 56
SparkFreqEncoderTransformer (*class in sparklightautoml.transformers.categorical*), 58
SparkHoldoutIterator (*class in sparklightautoml.validation.iterators*), 66
SparkImportanceEstimator (*class in sparklightautoml.pipelines.selection.base*), 27
SparkLabelEncoderEstimator (*class in sparklightautoml.transformers.categorical*), 55
SparkLabelEncoderTransformer (*class in sparklightautoml.transformers.categorical*), 58
SparkLGBAdvancedPipeline (*class in sparklightautoml.pipelines.features.lgb_pipeline*), 34
SparkLGBSimpleFeatures (*class in sparklightautoml.pipelines.features.lgb_pipeline*), 34
SparkLinearFeatures (*class in sparklightautoml.pipelines.features.linear_pipeline*), 35
SparkLinearLBFGS (*class in sparklightautoml.ml_algo.linear_pyspark*), 21
SparkLogOddsTransformer (*class in sparklightautoml.transformers.numeric*), 53
SparkMeanBlender (*class in sparklightautoml.automl.blend*), 12
SparkMetric (*class in sparklightautoml.tasks.base*), 47
SparkMLPipeline (*class in sparklightautoml.pipelines.ml.base*), 37
SparkMulticlassTargetEncoderEstimator (*class in sparklightautoml.transformers.categorical*), 56
SparkMultiTargetEncoderTransformer (*class in sparklightautoml.transformers.categorical*), 59
SparkNaNFlagsEstimator (*class in sparklightautoml.transformers.numeric*), 52
SparkNaNFlagsTransformer (*class in sparklightautoml.transformers.numeric*), 54
SparkNestedTabularMLPipeline (*class in sparklightautoml.pipelines.ml.nested_ml_pipe*), 38
SparkNoOpTransformer (*class in sparklightautoml.pipelines.features.base*), 33
SparkNpPermutationImportanceEstimator (*class in sparklightautoml.pipelines.selection.permutation_importance_based*), 27
SparkOHEEncoderEstimator (*class in sparklightautoml.transformers.categorical*), 57
SparkOrdinalEncoderEstimator (*class in sparklightautoml.transformers.categorical*), 55
SparkOrdinalEncoderTransformer (*class in sparklightautoml.transformers.categorical*), 58
SparkQuantileBinningEstimator (*class in sparklightautoml.transformers.numeric*), 53
SparkQuantileBinningTransformer (*class in sparklightautoml.transformers.numeric*), 54
SparkReaderHelper (*class in sparklightautoml.reader.base*), 41
SparkSequentialTransformer (*class in sparklightautoml.transformers.base*), 50
SparkStandardScalerEstimator (*class in sparklightautoml.transformers.numeric*), 53
SparkStandardScalerTransformer (*class in sparklightautoml.transformers.numeric*), 54
SparkTabularAutoML (*class in sparklightautoml.automl.presets.tabular_presets*), 7
SparkTabularDataFeatures (*class in sparklightautoml.pipelines.features.base*), 30
SparkTabularMLAlgo (*class in sparklightautoml.ml_algo.base*), 19
SparkTargetEncoderEstimator (*class in sparklightautoml.transformers.categorical*), 56
SparkTask (*class in sparklightautoml.tasks.base*), 47
SparkTimeToNumTransformer (*class in sparklightautoml.transformers.datetime*), 63
SparkToSparkReader (*class in sparklightautoml.reader.base*), 39
SparkToSparkReaderTransformer (*class in sparklightautoml.reader.base*), 41
SparkUnionTransformer (*class in sparklightautoml.transformers.base*), 50
SparkWeightedBlender (*class in sparklightautoml.automl.blend*), 12
split_models() (*sparklightautoml.automl.blend.SparkBlender method*), 11

T

transformer() (*sparklightautoml.automl.blend.SparkBlender method*), 11

U

unpersist() (*sparklightautoml.dataset.base.SparkDataset method*), 14
Unpersistable (*class in sparklightautoml.dataset.base*), 15