# MatchZoo Documentation

*Release 2.1*

**MatchZoo**

**May 19, 2019**

# Contents:

MatchZoo is a toolkit for text matching. It was developed with a focus on facilitating the designing, comparing and sharing of deep text matching models. There are a number of deep matching methods, such as DRMM, MatchPyramid, MV-LSTM, aNMM, DUET, ARC-I, ARC-II, DSSM, and CDSSM, designed with a unified interface. Potential tasks related to MatchZoo include document retrieval, question answering, conversational response ranking, paraphrase identification, etc. We are always happy to receive any code contributions, suggestions, comments from all our MatchZoo users.

CHAPTER 1

# matchzoo

## 1.1 matchzoo package

### 1.1.1 Subpackages

#### matchzoo.auto package

#### Subpackages

#### matchzoo.auto.preparer package

#### Submodules

#### matchzoo.auto.preparer.prepare module

matchzoo.auto.preparer.prepare.**prepare**(*task*, *model_class*, *data_pack*, *preprocessor=None*, *embedding=None*, *config=None*)

A simple shorthand for using `matchzoo.Preparer`.

*config* is used to control specific behaviors. The default *config* will be updated accordingly if a *config* dictionary is passed. e.g. to override the default *bin_size*, pass *config={'bin_size': 15}*.

**Parameters**

- **task** (*BaseTask*) – Task.
- **model_class** (Type[*BaseModel*]) – Model class.
- **data_pack** (*DataPack*) – DataPack used to fit the preprocessor.
- **preprocessor** (Optional[*BasePreprocessor*]) – Preprocessor used to fit the *data_pack*. (default: the default preprocessor of *model_class*)

- **embedding** (Optional[*Embedding*]) – Embedding to build a embedding matrix. If not set, then a correctly shaped randomized matrix will be built.
- **config** (Optional[dict]) – Configuration of specific behaviors. (default: return value of *mz.Preparer.get_default_config()*)

**Returns** A tuple of *(model, preprocessor, data_generator_builder, embedding_matrix)*.

### matchzoo.auto.preparer.preparer module

**class** matchzoo.auto.preparer.preparer.**Preparer**(*task*, *config=None*)

Bases: object

Unified setup processes of all MatchZoo models.

*config* is used to control specific behaviors. The default *config* will be updated accordingly if a *config* dictionary is passed. e.g. to override the default *bin_size*, pass *config={'bin_size': 15}*.

See *tutorials/automation.ipynb* for a detailed walkthrough on usage.

Default *config*:

{ # pair generator builder kwargs 'num_dup': 1,

# histogram unit of DRMM 'bin_size': 30, 'hist_mode': 'LCH',

# dynamic Pooling of MatchPyramid 'compress_ratio_left': 1.0, 'compress_ratio_right': 1.0,

# if no *matchzoo.Embedding* is passed to *tune* 'embedding_output_dim': 50

}

**Parameters**

- **task** (*BaseTask*) – Task.
- **config** (Optional[dict]) – Configuration of specific behaviors.

#### Example

```
>>> import matchzoo as mz
>>> task = mz.tasks.Ranking(loss=mz.losses.RankCrossEntropyLoss())
>>> preparer = mz.auto.Preparer(task)
>>> model_class = mz.models.DenseBaseline
>>> train_raw = mz.datasets.toy.load_data('train', 'ranking')
>>> model, prpr, gen_builder, matrix = preparer.prepare(model_class,
...                                                     train_raw)
>>> model.params.completed()
True
```

**classmethod get_default_config**()

Default config getter.

**Return type** dict

**prepare**(*model_class*, *data_pack*, *preprocessor=None*, *embedding=None*)

Prepare.

**Parameters**

- **model_class** (Type[*BaseModel*]) – Model class.

- **data_pack** (*DataPack*) – DataPack used to fit the preprocessor.
- **preprocessor** (Optional[*BasePreprocessor*]) – Preprocessor used to fit the *data_pack*. (default: the default preprocessor of *model_class*)
- **embedding** (Optional[*Embedding*]) – Embedding to build a embedding matrix. If not set, then a correctly shaped randomized matrix will be built.

**Return type** Tuple[*BaseModel*, *BasePreprocessor*, *DataGeneratorBuilder*, ndarray]

**Returns** A tuple of *(model, preprocessor, data_generator_builder, embedding_matrix)*.

### Module contents

### matchzoo.auto.tuner package

### Subpackages

### matchzoo.auto.tuner.callbacks package

### Submodules

### matchzoo.auto.tuner.callbacks.callback module

**class** matchzoo.auto.tuner.callbacks.callback.**Callback**

Bases: object

Tuner callback base class.

To build your own callbacks, inherit *mz.auto.tuner.callbacks.Callback* and overrides corresponding methods.

A run proceeds in the following way:

- run start (callback)
- build model
- build end (callback)
- fit and evaluate model
- collect result
- run end (callback)

This process is repeated for *num_runs* times in a tuner.

**on_build_end**(*tuner*, *model*)

Callback on build end stage.

**Parameters**

- **tuner** (*Tuner*) – Tuner.
- **model** (*BaseModel*) – A built model ready for fitting and evluating. Changes to this model affect the fitting and evaluating process.

**on_run_end**(*tuner*, *model*, *result*)

Callback on run end stage.

**Parameters**

- **tuner** (*Tuner*) – Tuner.
- **model** (*BaseModel*) – A built model done fitting and evaluating. Changes to the model will no longer affect the result.
- **result** (dict) – Result of the run. Changes to this dictionary will be visible in the return value of the *tune* method.

**on_run_start** (*tuner*, *sample*)
Callback on run start stage.

**Parameters**

- **tuner** (*Tuner*) – Tuner.
- **sample** (dict) – Sampled hyper space. Changes to this dictionary affects the model building process of the tuner.

### matchzoo.auto.tuner.callbacks.lambda_callback module

**class** matchzoo.auto.tuner.callbacks.lambda_callback.**LambdaCallback**(*on_run_start=None*, *on_build_end=None*, *on_run_end=None*)

Bases: *matchzoo.auto.tuner.callbacks.callback.Callback*

LambdaCallback. Just a shorthand for creating a callback class.

See matchzoo.tuner.callbacks.Callback for more details.

#### Example

```
>>> import matchzoo as mz
>>> model = mz.models.Naive()
>>> model.guess_and_fill_missing_params(verbose=0)
>>> data = mz.datasets.toy.load_data()
>>> data = model.get_default_preprocessor().fit_transform(
...     data, verbose=0)
>>> def show_inputs(*args):
...     print(' '.join(map(str, map(type, args))))
>>> callback = mz.auto.tuner.callbacks.LambdaCallback(
...     on_run_start=show_inputs,
...     on_build_end=show_inputs,
...     on_run_end=show_inputs
... )
>>> _ = mz.auto.tune(
...     params=model.params,
...     train_data=data,
...     test_data=data,
...     num_runs=1,
...     callbacks=[callback],
...     verbose=0,
... ) # noqa: E501
<class 'matchzoo.auto.tuner.tuner.Tuner'> <class 'dict'>
<class 'matchzoo.auto.tuner.tuner.Tuner'> <class 'matchzoo.models.naive.Naive'>
<class 'matchzoo.auto.tuner.tuner.Tuner'> <class 'matchzoo.models.naive.Naive'>
↪<class 'dict'>
```

**on_build_end**(*tuner*, *model*)
*on_build_end*.

**on_run_end**(*tuner*, *model*, *result*)
*on_run_end*.

**on_run_start**(*tuner*, *sample*)
*on_run_start*.

### matchzoo.auto.tuner.callbacks.load_embedding_matrix module

**class** matchzoo.auto.tuner.callbacks.load_embedding_matrix.**LoadEmbeddingMatrix**(*embedding_matrix*
Bases: *matchzoo.auto.tuner.callbacks.callback.Callback*

Load a pre-trained embedding after the model is built.

Used with tuner to load a pre-trained embedding matrix for each newly built model instance.

**Parameters** **embedding_matrix** – Embedding matrix to load.

#### Example

```
>>> import matchzoo as mz
>>> model = mz.models.ArcI()
>>> prpr = model.get_default_preprocessor()
>>> data = mz.datasets.toy.load_data()
>>> data = prpr.fit_transform(data, verbose=0)
>>> embed = mz.datasets.toy.load_embedding()
>>> term_index = prpr.context['vocab_unit'].state['term_index']
>>> matrix = embed.build_matrix(term_index)
>>> callback = mz.auto.tuner.callbacks.LoadEmbeddingMatrix(matrix)
>>> model.params.update(prpr.context)
>>> model.params['task'] = mz.tasks.Ranking()
>>> model.params['embedding_output_dim'] = embed.output_dim
>>> result = mz.auto.tune(
...     params=model.params,
...     train_data=data,
...     test_data=data,
...     num_runs=1,
...     callbacks=[callback],
...     verbose=0
... )
```

**on_build_end**(*tuner*, *model*)
*on_build_end*.

### matchzoo.auto.tuner.callbacks.save_model module

**class** matchzoo.auto.tuner.callbacks.save_model.**SaveModel**(*dir_path=PosixPath('/home/docs/.matchzoo/tunea*
Bases: *matchzoo.auto.tuner.callbacks.callback.Callback*

Save trained model.

For each trained model, a UUID will be generated as the *model_id*, the model will be saved under the *dir_path/model_id*. A *model_id* key will also be inserted into the result, which will visible in the return value of the *tune* method.

**Parameters** **dir_path** (Union[str, Path]) – Path to save the models to. (default: *matchzoo.USER_TUNED_MODELS_DIR*)

**on_run_end**(*tuner*, *model*, *result*)
Save model on run end.

### Module contents

### Submodules

### matchzoo.auto.tuner.tune module

matchzoo.auto.tuner.tune.**tune**(*params*, *train_data*, *test_data*, *fit_kwargs=None*, *evaluate_kwargs=None*, *metric=None*, *mode='maximize'*, *num_runs=10*, *callbacks=None*, *verbose=1*)

Tune model hyper-parameters.

A simple shorthand for using matchzoo.auto.Tuner.

*model.params.hyper_space* reprensents the model's hyper-parameters search space, which is the cross-product of individual hyper parameter's hyper space. When a *Tuner* builds a model, for each hyper parameter in *model.params*, if the hyper-parameter has a hyper-space, then a sample will be taken in the space. However, if the hyper-parameter does not have a hyper-space, then the default value of the hyper-parameter will be used.

See *tutorials/model_tuning.ipynb* for a detailed walkthrough on usage.

**Parameters**

- **params** (*ParamTable*) – A completed parameter table to tune. Usually *model.params* of the desired model to tune. *params.completed()* should be *True*.
- **train_data** (Union[*DataPack*, *DataGenerator*]) – Training data to use. Either a preprocessed *DataPack*, or a *DataGenerator*.
- **test_data** (Union[*DataPack*, *DataGenerator*]) – Testing data to use. A preprocessed *DataPack*.
- **fit_kwargs** (Optional[dict]) – Extra keyword arguments to pass to *fit*. (default: *dict(epochs=10, verbose=0)*)
- **evaluate_kwargs** (Optional[dict]) – Extra keyword arguments to pass to *evaluate*.
- **metric** (Union[str, *BaseMetric*, None]) – Metric to tune upon. Must be one of the metrics in *model.params['task'].metrics*. (default: the first metric in *params.['task'].metrics*.
- **mode** (str) – Either *maximize* the metric or *minimize* the metric. (default: 'maximize')
- **num_runs** (int) – Number of runs. Each run takes a sample in *params.hyper_space* and build a model based on the sample. (default: 10)
- **callbacks** (Optional[List[*Callback*]]) – A list of callbacks to handle. Handled sequentially at every callback point.
- **verbose** – Verbosity. (default: 1)

#### Example

```
>>> import matchzoo as mz
>>> train = mz.datasets.toy.load_data('train')
>>> dev = mz.datasets.toy.load_data('dev')
>>> prpr = mz.models.DenseBaseline.get_default_preprocessor()
>>> train = prpr.fit_transform(train, verbose=0)
>>> dev = prpr.transform(dev, verbose=0)
>>> model = mz.models.DenseBaseline()
>>> model.params['input_shapes'] = prpr.context['input_shapes']
>>> model.params['task'] = mz.tasks.Ranking()
>>> results = mz.auto.tune(
...     params=model.params,
...     train_data=train,
...     test_data=dev,
...     num_runs=1,
...     verbose=0
... )
>>> sorted(results['best'].keys())
['#', 'params', 'sample', 'score']
```

### matchzoo.auto.tuner.tuner module

**class** matchzoo.auto.tuner.tuner.**Tuner**(*params*, *train_data*, *test_data*, *fit_kwargs=None*, *evaluate_kwargs=None*, *metric=None*, *mode='maximize'*, *num_runs=10*, *callbacks=None*, *verbose=1*)

Bases: object

Model hyper-parameters tuner.

*model.params.hyper_space* reprensents the model's hyper-parameters search space, which is the cross-product of individual hyper parameter's hyper space. When a *Tuner* builds a model, for each hyper parameter in *model.params*, if the hyper-parameter has a hyper-space, then a sample will be taken in the space. However, if the hyper-parameter does not have a hyper-space, then the default value of the hyper-parameter will be used.

See *tutorials/model_tuning.ipynb* for a detailed walkthrough on usage.

**param params** A completed parameter table to tune. Usually *model.params* of the desired model to tune. *params.completed()* should be *True*.

**param train_data** Training data to use. Either a preprocessed *DataPack*, or a *DataGenerator*.

**param test_data** Testing data to use. A preprocessed *DataPack*.

**param fit_kwargs** Extra keyword arguments to pass to *fit*. (default: *dict(epochs=10, verbose=0)*)

**param evaluate_kwargs** Extra keyword arguments to pass to *evaluate*.

**param metric** Metric to tune upon. Must be one of the metrics in *model.params['task'].metrics*. (default: the first metric in *params.['task'].metrics*.

**param mode** Either *maximize* the metric or *minimize* the metric. (default: 'maximize')

**param num_runs** Number of runs. Each run takes a sample in *params.hyper_space* and build a model based on the sample. (default: 10)

**param callbacks** A list of callbacks to handle. Handled sequentially at every callback point.

**param verbose** Verbosity. (default: 1)

**Example:**

```
>>> import matchzoo as mz
>>> train = mz.datasets.toy.load_data('train')
>>> dev = mz.datasets.toy.load_data('dev')
>>> prpr = mz.models.DenseBaseline.get_default_preprocessor()
>>> train = prpr.fit_transform(train, verbose=0)
>>> dev = prpr.transform(dev, verbose=0)
>>> model = mz.models.DenseBaseline()
>>> model.params['input_shapes'] = prpr.context['input_shapes']
>>> model.params['task'] = mz.tasks.Ranking()
>>> tuner = mz.auto.Tuner(
...     params=model.params,
...     train_data=train,
...     test_data=dev,
...     num_runs=1,
...     verbose=0
... )
>>> results = tuner.tune()
>>> sorted(results['best'].keys())
['#', 'params', 'sample', 'score']
```

**callbacks**
*callbacks* getter.

**evaluate_kwargs**
*evaluate_kwargs* getter.

**fit_kwargs**
*fit_kwargs* getter.

**metric**
*metric* getter.

**mode**
*mode* getter.

**num_runs**
*num_runs* getter.

**params**
*params* getter.

**test_data**
*test_data* getter.

**train_data**
*train_data* getter.

**tune**()
Start tuning.

Notice that *tune* does not affect the tuner's inner state, so each new call to *tune* starts fresh. In other words, hyperspaces are suggestive only within the same *tune* call.

**verbose**
*verbose* getter.

### Module contents

### Module contents

### matchzoo.data_generator package

### Subpackages

### matchzoo.data_generator.callbacks package

### Submodules

### matchzoo.data_generator.callbacks.callback module

**class** matchzoo.data_generator.callbacks.callback.**Callback**
Bases: object

DataGenerator callback base class.

To build your own callbacks, inherit *mz.data_generator.callbacks.Callback* and overrides corresponding methods.

A batch is processed in the following way:

- slice data pack based on batch index
- handle *on_batch_data_pack* callbacks
- unpack data pack into x, y
- handle *on_batch_x_y* callbacks
- return x, y

**on_batch_data_pack**(*data_pack*)
*on_batch_data_pack*.

**Parameters** **data_pack** (*DataPack*) – a sliced DataPack before unpacking.

**on_batch_unpacked**(*x*, *y*)
*on_batch_unpacked*.

**Parameters**

- **x** (dict) – unpacked x.
- **y** (ndarray) – unpacked y.

### matchzoo.data_generator.callbacks.dynamic_pooling module

**class** matchzoo.data_generator.callbacks.dynamic_pooling.**DynamicPooling**(*fixed_length_left*, *fixed_length_right*, *compress_ratio_left=1*, *compress_ratio_right=1*)

Bases: *matchzoo.data_generator.callbacks.callback.Callback*

DPoolPairDataGenerator constructor.

**Parameters**

- **fixed_length_left** (int) – max length of left text.
- **fixed_length_right** (int) – max length of right text.
- **compress_ratio_left** (float) – the length change ratio, especially after normal pooling layers.
- **compress_ratio_right** (float) – the length change ratio, especially after normal pooling layers.

**on_batch_unpacked**(*x*, *y*)

Insert *dpool_index* into *x*.

**Parameters**

- **x** – unpacked x.
- **y** – unpacked y.

### matchzoo.data_generator.callbacks.histogram module

**class** matchzoo.data_generator.callbacks.histogram.**Histogram**(*embedding_matrix*, *bin_size=30*, *hist_mode='CH'*)

Bases: *matchzoo.data_generator.callbacks.callback.Callback*

Generate data with matching histogram.

**Parameters**

- **embedding_matrix** (ndarray) – The embedding matrix used to generator match histogram.
- **bin_size** (int) – The number of bin size of the histogram.
- **hist_mode** (str) – The mode of the MatchingHistogramUnit, one of *CH*, *NH*, and *LCH*.

**on_batch_unpacked**(*x*, *y*)

Insert *match_histogram* to *x*.

### matchzoo.data_generator.callbacks.lambda_callback module

**class** matchzoo.data_generator.callbacks.lambda_callback.**LambdaCallback**(*on_batch_data_pack=None*, *on_batch_unpacked=None*)

Bases: *matchzoo.data_generator.callbacks.callback.Callback*

LambdaCallback. Just a shorthand for creating a callback class.

See `matchzoo.data_generator.callbacks.Callback` for more details.

#### Example

```
>>> import matchzoo as mz
>>> from matchzoo.data_generator.callbacks import LambdaCallback
>>> data = mz.datasets.toy.load_data()
>>> batch_func = lambda x: print(type(x))
>>> unpack_func = lambda x, y: print(type(x), type(y))
>>> callback = LambdaCallback(on_batch_data_pack=batch_func,
...                           on_batch_unpacked=unpack_func)
>>> data_gen = mz.DataGenerator(
...     data, batch_size=len(data), callbacks=[callback])
>>> _ = data_gen[0]
<class 'matchzoo.data_pack.data_pack.DataPack'>
<class 'dict'> <class 'numpy.ndarray'>
```

**on_batch_data_pack**(*data_pack*)
*on_batch_data_pack*.

**on_batch_unpacked**(*x*, *y*)
*on_batch_unpacked*.

### Module contents

### Submodules

### matchzoo.data_generator.data_generator module

Base generator.

**class** matchzoo.data_generator.data_generator.**DataGenerator**(*data_pack*, *mode='point'*, *num_dup=1*, *num_neg=1*, *resample=True*, *batch_size=128*, *shuffle=True*, *callbacks=None*)

Bases: `keras.utils.data_utils.Sequence`

Data Generator.

Used to divide a `matchzoo.DataPack` into batches. This is helpful for generating batch-wise features and delaying data preprocessing to the *fit* time.

See *tutorials/data_handling.ipynb* for a walkthrough.

**Parameters**

- **data_pack** (*DataPack*) – DataPack to generator data from.
- **mode** – One of "point", "pair", and "list". (default: "point")
- **num_dup** (`int`) – Number of duplications per instance, only effective when *mode* is "pair". (default: 1)

- **num_neg** (int) – Number of negative samples per instance, only effective when *mode* is "pair". (default: 1)
- **resample** (bool) – Either to resample for each epoch, only effective when *mode* is "pair". (default: *True*)
- **batch_size** (int) – Batch size. (default: 128)
- **shuffle** (bool) – Either to shuffle the samples/instances. (default: *True*)
- **callbacks** (Optional[List[*Callback*]]) – Callbacks. See *matchzoo.data_generator.callbacks* for more details.

**Examples::**

```
>>> import numpy as np
>>> import matchzoo as mz
>>> np.random.seed(0)
>>> data_pack = mz.datasets.toy.load_data()
>>> batch_size = 8
```

**To generate data points:**

```
>>> point_gen = mz.DataGenerator(
...     data_pack=data_pack,
...     batch_size=batch_size
... )
>>> len(point_gen)
13
>>> x, y = point_gen[0]
>>> for key, value in sorted(x.items()):
...     print(key, str(value)[:30])
id_left ['Q6' 'Q17' 'Q1' 'Q13' 'Q16' '
id_right ['D6-6' 'D17-1' 'D1-2' 'D13-3'
text_left ['how long is the term for fed
text_right ['See Article I and Article II
```

**To generate data pairs:**

```
>>> pair_gen = mz.DataGenerator(
...     data_pack=data_pack,
...     mode='pair',
...     num_dup=4,
...     num_neg=4,
...     batch_size=batch_size,
...     shuffle=False
... )
>>> len(pair_gen)
3
>>> x, y = pair_gen[0]
>>> for key, value in sorted(x.items()):
...     print(key, str(value)[:30])
id_left ['Q1' 'Q1' 'Q1' 'Q1' 'Q1' 'Q1'
id_right ['D1-3' 'D1-4' 'D1-0' 'D1-1' '
text_left ['how are glacier caves formed
text_right ['A glacier cave is a cave for
```

**To generate data lists:** # TODO:

**batch_indices**
*batch_indices* getter.

**batch_size**
*batch_size* getter.

**callbacks**
*callbacks* getter.

**mode**
*mode* getter.

**num_dup**
*num_dup* getter.

**num_neg**
*num_neg* getter.

**on_epoch_end**()
Reorganize the index array while epoch is ended.

**reset_index**()
Set the `index_array`.

Here the `index_array` records the index of all the instances.

**shuffle**
*shuffle* getter.

### matchzoo.data_generator.data_generator_builder module

**class** matchzoo.data_generator.data_generator_builder.**DataGeneratorBuilder**(***kwargs*)
Bases: `object`

Data Generator Bulider. In essense a wrapped partial function.

#### Example

```
>>> import matchzoo as mz
>>> builder = mz.DataGeneratorBuilder(mode='pair', batch_size=32)
>>> data = mz.datasets.toy.load_data()
>>> gen = builder.build(data)
>>> type(gen)
<class 'matchzoo.data_generator.data_generator.DataGenerator'>
>>> gen.batch_size
32
>>> gen_64 = builder.build(data, batch_size=64)
>>> gen_64.batch_size
64
```

**build**(*data_pack*, ***kwargs*)
Build a DataGenerator.

**Parameters**

- **data_pack** – DataPack to build upon.
- **kwargs** – Additional keyword arguments to override the keyword arguments passed in *__init__*.

**Return type** *DataGenerator*

### Module contents

### matchzoo.data_pack package

### Submodules

### matchzoo.data_pack.data_pack module

Matchzoo DataPack, pair-wise tuple (feature) and context as input.

**class** matchzoo.data_pack.data_pack.**DataPack**(*relation*, *left*, *right*)

Bases: object

Matchzoo *DataPack* data structure, store dataframe and context.

*DataPack* is a MatchZoo native data structure that most MatchZoo data handling processes build upon. A *DataPack* consists of three parts: *left*, *right* and *relation*, each one of is a *pandas.DataFrame*.

**Parameters**

- **relation** (DataFrame) – Store the relation between left document and right document use ids.
- **left** (DataFrame) – Store the content or features for id_left.
- **right** (DataFrame) – Store the content or features for id_right.

#### Example

```
>>> left = [
...     ['qid1', 'query 1'],
...     ['qid2', 'query 2']
... ]
>>> right = [
...     ['did1', 'document 1'],
...     ['did2', 'document 2']
... ]
>>> relation = [['qid1', 'did1', 1], ['qid2', 'did2', 1]]
>>> relation_df = pd.DataFrame(relation)
>>> left = pd.DataFrame(left)
>>> right = pd.DataFrame(right)
>>> dp = DataPack(
...     relation=relation_df,
...     left=left,
...     right=right,
... )
>>> len(dp)
2
```

**DATA_FILENAME = 'data.dill'**

**class FrameView**(*data_pack*)

Bases: object

FrameView.

**append_text_length**(*verbose=1*)

Append *length_left* and *length_right* columns.

**Parameters**

- **inplace** – *True* to modify inplace, *False* to return a modified copy. (default: *False*)
- **verbose** – Verbosity.

### Example

```
>>> import matchzoo as mz
>>> data_pack = mz.datasets.toy.load_data()
>>> 'length_left' in data_pack.frame[0].columns
False
>>> new_data_pack = data_pack.append_text_length(verbose=0)
>>> 'length_left' in new_data_pack.frame[0].columns
True
>>> 'length_left' in data_pack.frame[0].columns
False
>>> data_pack.append_text_length(inplace=True, verbose=0)
>>> 'length_left' in data_pack.frame[0].columns
True
```

**apply_on_text**(*func*, *mode='both'*, *rename=None*, *verbose=1*)

Apply *func* to text columns based on *mode*.

**Parameters**

- **func** (`Callable`) – The function to apply.
- **mode** (`str`) – One of "both", "left" and "right".
- **rename** (`Optional[str]`) – If set, use new names for results instead of replacing the original columns. To set *rename* in "both" mode, use a tuple of *str*, e.g. ("text_left_new_name", "text_right_new_name").
- **inplace** – *True* to modify inplace, *False* to return a modified copy. (default: *False*)
- **verbose** (`int`) – Verbosity.

**Examples::**

```
>>> import matchzoo as mz
>>> data_pack = mz.datasets.toy.load_data()
>>> frame = data_pack.frame
```

**To apply *len* on the left text and add the result as 'length_left':**

```
>>> data_pack.apply_on_text(len, mode='left',
...                         rename='length_left',
...                         inplace=True,
...                         verbose=0)
>>> list(frame[0].columns)  # noqa: E501
['id_left', 'text_left', 'length_left', 'id_right', 'text_right', 'label']
```

**To do the same to the right text:**

```
>>> data_pack.apply_on_text(len, mode='right',
...                         rename='length_right',
...                         inplace=True,
...                         verbose=0)
>>> list(frame[0].columns)  # noqa: E501
['id_left', 'text_left', 'length_left', 'id_right', 'text_right', 'length_
↪right', 'label']
```

**To do the same to the both texts at the same time:**

```
>>> data_pack.apply_on_text(len, mode='both',
...                         rename=('extra_left', 'extra_right'),
...                         inplace=True,
...                         verbose=0)
>>> list(frame[0].columns)  # noqa: E501
['id_left', 'text_left', 'length_left', 'extra_left', 'id_right', 'text_
↪right', 'length_right', 'extra_right', 'label']
```

**To suppress outputs:**

```
>>> data_pack.apply_on_text(len, mode='both', verbose=0,
...                         inplace=True)
```

**copy**()

**Return type** *DataPack*

**Returns** A deep copy.

**drop_label**()

Remove *label* column from the data pack.

**Parameters** **inplace** – *True* to modify inplace, *False* to return a modified copy. (default: *False*)

### Example

```
>>> import matchzoo as mz
>>> data_pack = mz.datasets.toy.load_data()
>>> data_pack.has_label
True
>>> data_pack.drop_label(inplace=True)
>>> data_pack.has_label
False
```

**frame**

View the data pack as a `pandas.DataFrame`.

Returned data frame is created by merging the left data frame, the right dataframe and the relation data frame. Use *[]* to access an item or a slice of items.

**Return type** *FrameView*

**Returns** A `matchzoo.DataPack.FrameView` instance.

### Example

```
>>> import matchzoo as mz
>>> data_pack = mz.datasets.toy.load_data()
>>> type(data_pack.frame)
<class 'matchzoo.data_pack.data_pack.DataPack.FrameView'>
>>> frame_slice = data_pack.frame[0:5]
>>> type(frame_slice)
<class 'pandas.core.frame.DataFrame'>
>>> list(frame_slice.columns)
['id_left', 'text_left', 'id_right', 'text_right', 'label']
>>> full_frame = data_pack.frame()
>>> len(full_frame) == len(data_pack)
True
```

**has_label**
*True* if *label* column exists, *False* other wise.

**Type** return

**Return type** bool

**left**
Get *left()* of *DataPack*.

**Return type** DataFrame

**one_hot_encode_label**(*num_classes=2*)
One-hot encode *label* column of *relation*.

**Parameters**

- **num_classes** – Number of classes.
- **inplace** – *True* to modify inplace, *False* to return a modified copy. (default: *False*)

**Returns**

**relation**
*relation* getter.

**right**
Get *right()* of *DataPack*.

**Return type** DataFrame

**save**(*dirpath*)
Save the *DataPack* object.

A saved *DataPack* is represented as a directory with a *DataPack* object (transformed user input as features and context), it will be saved by *pickle*.

**Parameters** **dirpath** (Union[str, Path]) – directory path of the saved *DataPack*.

**shuffle**()
Shuffle the data pack by shuffling the relation column.

**Parameters** **inplace** – *True* to modify inplace, *False* to return a modified copy. (default: *False*)

#### Example

```
>>> import matchzoo as mz
>>> import numpy.random
>>> numpy.random.seed(0)
>>> data_pack = mz.datasets.toy.load_data()
>>> orig_ids = data_pack.relation['id_left']
>>> shuffled = data_pack.shuffle()
>>> (shuffled.relation['id_left'] != orig_ids).any()
True
```

**unpack**()

Unpack the data for training.

The return value can be directly feed to *model.fit* or *model.fit_generator*.

**Return type** Tuple[Dict[str, <built-in function array>], Optional[<built-in function array>]]

**Returns** A tuple of (X, y). *y* is *None* if *self* has no label.

#### Example

```
>>> import matchzoo as mz
>>> data_pack = mz.datasets.toy.load_data()
>>> X, y = data_pack.unpack()
>>> type(X)
<class 'dict'>
>>> sorted(X.keys())
['id_left', 'id_right', 'text_left', 'text_right']
>>> type(y)
<class 'numpy.ndarray'>
>>> X, y = data_pack.drop_label().unpack()
>>> type(y)
<class 'NoneType'>
```

matchzoo.data_pack.data_pack.**load_data_pack**(*dirpath*)

Load a *DataPack*. The reverse function of save().

**Parameters** **dirpath** (Union[str, Path]) – directory path of the saved model.

**Return type** *DataPack*

**Returns** a *DataPack* instance.

### matchzoo.data_pack.pack module

Convert list of input into class:*DataPack* expected format.

matchzoo.data_pack.pack.**pack**(*df*)

Pack a DataPack using *df*.

The *df* must have *text_left* and *text_right* columns. Optionally, the *df* can have *id_left*, *id_right* to index *text_left* and *text_right* respectively. *id_left*, *id_right* will be automatically generated if not specified.

**Parameters** **df** (DataFrame) – Input pandas.DataFrame to use.

**Examples::**

```
>>> import matchzoo as mz
>>> import pandas as pd
>>> df = pd.DataFrame(data={'text_left': list('AABC'),
...                         'text_right': list('abbc'),
...                         'label': [0, 1, 1, 0]})
>>> mz.pack(df).frame()
  id_left text_left id_right text_right  label
0     L-0         A      R-0          a      0
1     L-0         A      R-1          b      1
2     L-1         B      R-1          b      1
3     L-2         C      R-2          c      0
```

**Return type** *DataPack*

### Module contents

### matchzoo.datasets package

### Subpackages

### matchzoo.datasets.embeddings package

### Submodules

### matchzoo.datasets.embeddings.load_glove_embedding module

Embedding data loader.

matchzoo.datasets.embeddings.load_glove_embedding.**load_glove_embedding**(*dimension=50*)

Return the pretrained glove embedding.

**Parameters** **dimension** (int) – the size of embedding dimension, the value can only be 50, 100, or 300.

**Return type** *Embedding*

**Returns** The mz.embedding.Embedding object.

### Module contents

### matchzoo.datasets.quora_qp package

### Submodules

### matchzoo.datasets.quora_qp.load_data module

Quora Question Pairs data loader.

matchzoo.datasets.quora_qp.load_data.**load_data**(*stage='train'*, *task='classification'*, *return_classes=False*)

Load QuoraQP data.

**Parameters**

- **path** – *None* for download from quora, specific path for downloaded data.
- **stage** (str) – One of *train*, *dev*, and *test*.
- **task** (str) – Could be one of *ranking*, *classification* or a matchzoo.engine.BaseTask instance.
- **return_classes** (bool) – Whether return classes for classification task.

**Return type** Union[*DataPack*, tuple]

**Returns** A DataPack if *ranking*, a tuple of (DataPack, classes) if *classification*.

### Module contents

### matchzoo.datasets.snli package

### Submodules

### matchzoo.datasets.snli.load_data module

SNLI data loader.

matchzoo.datasets.snli.load_data.**load_data**(*stage='train'*, *task='classification'*, *target_label='entailment'*, *return_classes=False*)

Load SNLI data.

**Parameters**

- **stage** (str) – One of *train*, *dev*, and *test*. (default: *train*)
- **task** (str) – Could be one of *ranking*, *classification* or a matchzoo.engine.BaseTask instance. (default: *ranking*)
- **target_label** (str) – If *ranking*, chose one of *entailment*, *contradiction*, *neutral*, and - as the positive label. (default: *entailment*)
- **return_classes** (bool) – *True* to return classes for classification task, *False* otherwise.

**Return type** Union[*DataPack*, tuple]

**Returns** A DataPack unless *task* is *classificiation* and *return_classes* is *True*: a tuple of *(DataPack, classes)* in that case.

### Module contents

### matchzoo.datasets.toy package

### Module contents

matchzoo.datasets.toy.**load_data**(*stage='train'*, *task='ranking'*, *return_classes=False*)

Load WikiQA data.

**Parameters**

- **stage** (str) – One of *train*, *dev*, and *test*.

- **task** (str) – Could be one of *ranking*, *classification* or a matchzoo.engine.BaseTask instance.
- **return_classes** (bool) – *True* to return classes for classification task, *False* otherwise.

**Return type** Union[*DataPack*, Tuple[*DataPack*, list]]

**Returns** A DataPack unless *task* is *classificiation* and *return_classes* is *True*: a tuple of *(DataPack, classes)* in that case.

#### Example

```
>>> import matchzoo as mz
>>> stages = 'train', 'dev', 'test'
>>> tasks = 'ranking', 'classification'
>>> for stage in stages:
...     for task in tasks:
...         _ = mz.datasets.toy.load_data(stage, task)
```

matchzoo.datasets.toy.**load_embedding**()

### matchzoo.datasets.wiki_qa package

#### Submodules

#### matchzoo.datasets.wiki_qa.load_data module

WikiQA data loader.

matchzoo.datasets.wiki_qa.load_data.**load_data**(*stage='train'*, *task='ranking'*, *filtered=False*, *return_classes=False*)

Load WikiQA data.

**Parameters**

- **stage** (str) – One of *train*, *dev*, and *test*.
- **task** (str) – Could be one of *ranking*, *classification* or a matchzoo.engine.BaseTask instance.
- **filtered** (bool) – Whether remove the questions without correct answers.
- **return_classes** (bool) – *True* to return classes for classification task, *False* otherwise.

**Return type** Union[*DataPack*, tuple]

**Returns** A DataPack unless *task* is *classificiation* and *return_classes* is *True*: a tuple of *(DataPack, classes)* in that case.

#### Module contents

#### Module contents

matchzoo.datasets.**list_available**()

## matchzoo.embedding package

### Submodules

### matchzoo.embedding.embedding module

Matchzoo toolkit for token embedding.

**class** matchzoo.embedding.embedding.**Embedding**(*data*)

Bases: object

Embedding class.

**Examples::**

```
>>> import matchzoo as mz
>>> train_raw = mz.datasets.toy.load_data()
>>> pp = mz.preprocessors.NaivePreprocessor()
>>> train = pp.fit_transform(train_raw, verbose=0)
>>> vocab_unit = mz.build_vocab_unit(train, verbose=0)
>>> term_index = vocab_unit.state['term_index']
>>> embed_path = mz.datasets.embeddings.EMBED_RANK
```

**To load from a file:**

```
>>> embedding = mz.embedding.load_from_file(embed_path)
>>> matrix = embedding.build_matrix(term_index)
>>> matrix.shape[0] == len(term_index)
True
```

**To build your own:**

```
>>> data = pd.DataFrame(data=[[0, 1], [2, 3]], index=['A', 'B'])
>>> embedding = mz.Embedding(data)
>>> matrix = embedding.build_matrix({'A': 2, 'B': 1, '_PAD': 0})
>>> matrix.shape == (3, 2)
True
```

**build_matrix**(*term_index*, *initializer=<function Embedding.<lambda>>*)

Build a matrix using *term_index*.

**Parameters**

- **term_index** (dict) – A *dict* or *TermIndex* to build with.
- **initializer** – A callable that returns a default value for missing terms in data. (default: a random uniform distribution in range) *(-0.2, 0.2)*).

**Return type** ndarray

**Returns** A matrix.

**input_dim**

return Embedding input dimension.

**Return type** int

**output_dim**

return Embedding output dimension.

**Return type** int

matchzoo.embedding.embedding.**load_from_file**(*file_path*, *mode='word2vec'*)
Load embedding from *file_path*.

**Parameters**

- **file_path** (str) – Path to file.
- **mode** (str) – Embedding file format mode, one of 'word2vec' or 'glove'. (default: 'word2vec')

**Return type** *Embedding*

**Returns** An matchzoo.embedding.Embedding instance.

### Module contents

### matchzoo.engine package

### Submodules

### matchzoo.engine.base_metric module

Metric base class and some related utilities.

**class** matchzoo.engine.base_metric.**BaseMetric**
Bases: abc.ABC

Metric base class.

**ALIAS = 'base_metric'**

matchzoo.engine.base_metric.**sort_and_couple**(*labels*, *scores*)
Zip the *labels* with *scores* into a single list.

**Return type** <built-in function array>

### matchzoo.engine.base_model module

Base Model.

**class** matchzoo.engine.base_model.**BaseModel**(*params=None*, *backend=None*)
Bases: abc.ABC

Abstract base class of all MatchZoo models.

MatchZoo models are wrapped over keras models, and the actual keras model built can be accessed by *model.backend*. *params* is a set of model hyper-parameters that deterministically builds a model. In other words, *params['model_class'](params=params)* of the same *params* always create models with the same structure.

**Parameters**

- **params** (Optional[*ParamTable*]) – Model hyper-parameters. (default: return value from *get_default_params()*)
- **backend** (Optional[Model]) – A keras model as the model backend. Usually not passed as an argument.

### Example

```
>>> BaseModel()  # doctest: +ELLIPSIS
Traceback (most recent call last):
...
TypeError: Can't instantiate abstract class BaseModel ...
>>> class MyModel(BaseModel):
...     def build(self):
...         pass
>>> isinstance(MyModel(), BaseModel)
True
```

**BACKEND_WEIGHTS_FILENAME = 'backend_weights.h5'**

**PARAMS_FILENAME = 'params.dill'**

**backend**
return model backend, a keras model instance.

**Return type** Model

**build**()
Build model, each subclass need to impelemnt this method.

**compile**()
Compile model for training.

Only *keras* native metrics are compiled together with backend. MatchZoo metrics are evaluated only through *evaluate()*. Notice that *keras* count *loss* as one of the metrics while MatchZoo matchzoo.engine.BaseTask does not.

### Examples

```
>>> from matchzoo import models
>>> model = models.Naive()
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.params['task'].metrics = ['mse', 'map']
>>> model.params['task'].metrics
['mse', mean_average_precision(0.0)]
>>> model.build()
>>> model.compile()
```

**evaluate**(*x*, *y*, *batch_size=128*)
Evaluate the model.

**Parameters**

- **x** (Dict[str, ndarray]) – Input data.
- **y** (ndarray) – Labels.
- **batch_size** (int) – Number of samples when *predict* for evaluation. (default: 128)

**Examples::**

```
>>> import matchzoo as mz
>>> data_pack = mz.datasets.toy.load_data()
>>> preprocessor = mz.preprocessors.NaivePreprocessor()
>>> data_pack = preprocessor.fit_transform(data_pack, verbose=0)
```

(continues on next page)

(continued from previous page)

```
>>> m = mz.models.DenseBaseline()
>>> m.params['task'] = mz.tasks.Ranking()
>>> m.params['task'].metrics = [
...     'acc', 'mse', 'mae', 'ce',
...     'average_precision', 'precision', 'dcg', 'ndcg',
...     'mean_reciprocal_rank', 'mean_average_precision', 'mrr',
...     'map', 'MAP',
...     mz.metrics.AveragePrecision(threshold=1),
...     mz.metrics.Precision(k=2, threshold=2),
...     mz.metrics.DiscountedCumulativeGain(k=2),
...     mz.metrics.NormalizedDiscountedCumulativeGain(
...         k=3, threshold=-1),
...     mz.metrics.MeanReciprocalRank(threshold=2),
...     mz.metrics.MeanAveragePrecision(threshold=3)
... ]
>>> m.guess_and_fill_missing_params(verbose=0)
>>> m.build()
>>> m.compile()
>>> x, y = data_pack.unpack()
>>> evals = m.evaluate(x, y)
>>> type(evals)
<class 'dict'>
```

**Return type** `Dict[BaseMetric, float]`

**evaluate_generator**(*generator*, *batch_size=128*)

Evaluate the model.

**Parameters**

- **generator** (*DataGenerator*) – DataGenerator to evluate.
- **batch_size** (`int`) – Batch size. (default: 128)

**Return type** `Dict[BaseMetric, float]`

**fit**(*x*, *y*, *batch_size=128*, *epochs=1*, *verbose=1*, ***kwargs*)

Fit the model.

See `keras.models.Model.fit()` for more details.

**Parameters**

- **x** (`Union[ndarray, List[ndarray], dict]`) – input data.
- **y** (`ndarray`) – labels.
- **batch_size** (`int`) – number of samples per gradient update.
- **epochs** (`int`) – number of epochs to train the model.
- **verbose** (`int`) – 0, 1, or 2. Verbosity mode. 0 = silent, 1 = verbose, 2 = one log line per epoch.

Key word arguments not listed above will be propagated to keras's fit.

**Return type** `History`

**Returns** A *keras.callbacks.History* instance. Its history attribute contains all information collected during training.

**fit_generator**(*generator*, *epochs=1*, *verbose=1*, ***kwargs*)
Fit the model with matchzoo *generator*.

See `keras.models.Model.fit_generator()` for more details.

**Parameters**

- **generator** (*DataGenerator*) – A generator, an instance of `engine.DataGenerator`.
- **epochs** (`int`) – Number of epochs to train the model.
- **verbose** (`int`) – 0, 1, or 2. Verbosity mode. 0 = silent, 1 = verbose, 2 = one log line per epoch.

**Return type** `History`

**Returns** A *keras.callbacks.History* instance. Its history attribute contains all information collected during training.

**classmethod get_default_params**(*with_embedding=False*, *with_multi_layer_perceptron=False*)
Model default parameters.

**The common usage is to instantiate `matchzoo.engine.ModelParams` first, then set the model specific parametrs.**

### Examples

```
>>> class MyModel(BaseModel):
...     def build(self):
...         print(self._params['num_eggs'], 'eggs')
...         print('and', self._params['ham_type'])
...
...     @classmethod
...     def get_default_params(cls):
...         params = ParamTable()
...         params.add(Param('num_eggs', 512))
...         params.add(Param('ham_type', 'Parma Ham'))
...         return params
>>> my_model = MyModel()
>>> my_model.build()
512 eggs
and Parma Ham
```

Notice that all parameters must be serialisable for the entire model to be serialisable. Therefore, it's strongly recommended to use python native data types to store parameters.

**Return type** *ParamTable*

**Returns** model parameters

**classmethod get_default_preprocessor**()
Model default preprocessor.

The preprocessor's transform should produce a correctly shaped data pack that can be used for training. Some extra configuration (e.g. setting *input_shapes* in `matchzoo.models.DSSMModel` may be required on the user's end.

**Return type** *BasePreprocessor*

**Returns** Default preprocessor.

**get_embedding_layer**(*name='embedding'*)
Get the embedding layer.

All MatchZoo models with a single embedding layer set the embedding layer name to *embedding*, and this method should return that layer.

**Parameters** **name** (str) – Name of the embedding layer. (default: *embedding*)

**Return type** Layer

**guess_and_fill_missing_params**(*verbose=1*)
Guess and fill missing parameters in *params*.

Use this method to automatically fill-in other hyper parameters. This involves some guessing so the parameter it fills could be wrong. For example, the default task is *Ranking*, and if we do not set it to *Classification* manaully for data packs prepared for classification, then the shape of the model output and the data will mismatch.

**Parameters** **verbose** – Verbosity.

**load_embedding_matrix**(*embedding_matrix*, *name='embedding'*)
Load an embedding matrix.

Load an embedding matrix into the model's embedding layer. The name of the embedding layer is specified by *name*. For models with only one embedding layer, set *name='embedding'* when creating the keras layer, and use the default *name* when load the matrix. For models with more than one embedding layers, initialize keras layer with different layer names, and set *name* accordingly to load a matrix to a chosen layer.

**Parameters**

- **embedding_matrix** (ndarray) – Embedding matrix to be loaded.
- **name** (str) – Name of the layer. (default: 'embedding')

**params**
model parameters.

**Type** return

**Return type** *ParamTable*

**predict**(*x*, *batch_size=128*)
Generate output predictions for the input samples.

See keras.models.Model.predict() for more details.

**Parameters**

- **x** (Dict[str, ndarray]) – input data
- **batch_size** – number of samples per gradient update

**Return type** ndarray

**Returns** numpy array(s) of predictions

**save**(*dirpath*)
Save the model.

A saved model is represented as a directory with two files. One is a model parameters file saved by *pickle*, and the other one is a model h5 file saved by *keras*.

**Parameters** **dirpath** (Union[str, Path]) – directory path of the saved model

#### Example

```
>>> import matchzoo as mz
>>> model = mz.models.Naive()
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
>>> model.save('temp-model')
>>> import shutil
>>> shutil.rmtree('temp-model')
```

matchzoo.engine.base_model.**load_model**(*dirpath*)

Load a model. The reverse function of *BaseModel.save()*.

**Parameters** **dirpath** (Union[str, Path]) – directory path of the saved model

**Return type** *BaseModel*

**Returns** a *BaseModel* instance

#### Example

```
>>> import matchzoo as mz
>>> model = mz.models.Naive()
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
>>> model.save('my-model')
>>> model.params.keys() == mz.load_model('my-model').params.keys()
True
>>> import shutil
>>> shutil.rmtree('my-model')
```

### matchzoo.engine.base_preprocessor module

*BasePreprocessor* define input and ouutput for processors.

**class** matchzoo.engine.base_preprocessor.**BasePreprocessor**

Bases: object

*BasePreprocessor* to input handle data.

A preprocessor should be used in two steps. First, *fit*, then, *transform*. *fit* collects information into *context*, which includes everything the preprocessor needs to *transform* together with other useful information for later use. *fit* will only change the preprocessor's inner state but not the input data. In contrast, *transform* returns a modified copy of the input data without changing the preprocessor's inner state.

**DATA_FILENAME = 'preprocessor.dill'**

**context**

Return context.

**fit**(*data_pack*, *verbose=1*)

Fit parameters on input data.

This method is an abstract base method, need to be implemented in the child class.

This method is expected to return itself as a callable object.

**Parameters**

- **data_pack** (*DataPack*) – Datapack object to be fitted.
- **verbose** (int) – Verbosity.

**Return type** *BasePreprocessor*

**fit_transform**(*data_pack*, *verbose=1*)
Call fit-transform.

**Parameters**

- **data_pack** (*DataPack*) – DataPack object to be processed.
- **verbose** (int) – Verbosity.

**Return type** *DataPack*

**save**(*dirpath*)
Save the DSSMPreprocessor object.

A saved DSSMPreprocessor is represented as a directory with the *context* object (fitted parameters on training data), it will be saved by *pickle*.

**Parameters** **dirpath** (Union[str, Path]) – directory path of the saved DSSMPreprocessor.

**transform**(*data_pack*, *verbose=1*)
Transform input data to expected manner.

This method is an abstract base method, need to be implemented in the child class.

**Parameters**

- **data_pack** (*DataPack*) – DataPack object to be transformed.
- **verbose** (int) – Verbosity. or list of text-left, text-right tuples.

**Return type** *DataPack*

matchzoo.engine.base_preprocessor.**load_preprocessor**(*dirpath*)
Load the fitted *context*. The reverse function of save().

**Parameters** **dirpath** (Union[str, Path]) – directory path of the saved model.

**Return type** *DataPack*

**Returns** a DSSMPreprocessor instance.

matchzoo.engine.base_preprocessor.**validate_context**(*func*)
Validate context in the preprocessor.

### matchzoo.engine.base_task module

Base task.

**class** matchzoo.engine.base_task.**BaseTask**(*loss=None*, *metrics=None*)
Bases: abc.ABC

Base Task, shouldn't be used directly.

**classmethod list_available_losses**()

**Return type** list

**Returns** a list of available losses.

**classmethod list_available_metrics**()

**Return type** list

**Returns** a list of available metrics.

**loss**
Loss used in the task.

**Type** return

**metrics**
Metrics used in the task.

**Type** return

**output_dtype**
output data type for specific task.

**Type** return

**output_shape**
output shape of a single sample of the task.

**Type** return

**Return type** tuple

### matchzoo.engine.callbacks module

Callbacks.

**class** matchzoo.engine.callbacks.**EvaluateAllMetrics**(*model*, *x*, *y*, *once_every=1*, *batch_size=128*, *model_save_path=None*, *verbose=1*)

Bases: keras.callbacks.Callback

Callback to evaluate all metrics.

MatchZoo metrics can not be evaluated batch-wise since they require dataset-level information. As a result, MatchZoo metrics are not evaluated automatically when a Model *fit*. When this callback is used, all metrics, including MatchZoo metrics and Keras metrics, are evluated once every *once_every* epochs.

**Parameters**

- **model** (*BaseModel*) – Model to evaluate.
- **x** (Union[ndarray, List[ndarray]]) –

  X.
- **y** (ndarray) –

  y.
- **once_every** (int) – Evaluation only triggers when *epoch % once_every == 0*. (default: 1, i.e. evaluate on every epoch's end)
- **batch_size** (int) – Number of samples per evaluation. This only affects the evaluation of Keras metrics, since MatchZoo metrics are always evaluated using the full data.
- **model_save_path** (Optional[str]) – Directory path to save the model after each evaluate callback, (default: None, i.e., no saving.)

- **verbose** – Verbosity.

**on_epoch_end**(*epoch*, *logs=None*)
Called at the end of en epoch.

**Parameters**

- **epoch** (int) – integer, index of epoch.
- **logs** (Optional[dict]) – dictionary of logs.

**Returns** dictionary of logs.

### matchzoo.engine.hyper_spaces module

Hyper parameter search spaces wrapping *hyperopt*.

**class** matchzoo.engine.hyper_spaces.**HyperoptProxy**(*hyperopt_func*, ***kwargs*)
Bases: object

Hyperopt proxy class.

See *hyperopt*'s documentation for more details: https://github.com/hyperopt/hyperopt/wiki/FMin

Reason of these wrappers:

> A hyper space in *hyperopt* requires a *label* to instantiate. This *label* is used later as a reference to original hyper space that is sampled. In *matchzoo*, hyper spaces are used in matchzoo.engine.Param. Only if a hyper space's label matches its parent matchzoo.engine.Param's name, *matchzoo* can correctly back-refrenced the parameter got sampled. This can be done by asking the user always use the same name for a parameter and its hyper space, but typos can occur. As a result, these wrappers are created to hide hyper spaces' *label*, and always correctly bind them with its parameter's name.

**Examples::**

```
>>> import matchzoo as mz
>>> from hyperopt.pyll.stochastic import sample
```

**Basic Usage:**

```
>>> model = mz.models.DenseBaseline()
>>> sample(model.params.hyper_space)  # doctest: +SKIP
 {'mlp_num_layers': 1.0, 'mlp_num_units': 274.0}
```

**Arithmetic Operations:**

```
>>> new_space = 2 ** mz.hyper_spaces.quniform(2, 6)
>>> model.params.get('mlp_num_layers').hyper_space = new_space
>>> sample(model.params.hyper_space)  # doctest: +SKIP
{'mlp_num_layers': 8.0, 'mlp_num_units': 292.0}
```

**convert**(*name*)
Attach *name* as *hyperopt.hp*'s *label*.

**Parameters** **name** (str) –

**Return type** Apply

**Returns** a *hyperopt* ready search space

**class** matchzoo.engine.hyper_spaces.**choice**(*options*)
Bases: *matchzoo.engine.hyper_spaces.HyperoptProxy*

hyperopt.hp.choice() proxy.

**class** matchzoo.engine.hyper_spaces.**quniform**(*low*, *high*, *q=1*)
Bases: *matchzoo.engine.hyper_spaces.HyperoptProxy*

hyperopt.hp.quniform() proxy.

matchzoo.engine.hyper_spaces.**sample**(*space*)
Take a sample in the hyper space.

This method is stateless, so the distribution of the samples is different from that of *tune* call. This function just gives a general idea of what a sample from the *space* looks like.

#### Example

```
>>> import matchzoo as mz
>>> space = mz.models.Naive.get_default_params().hyper_space
>>> mz.hyper_spaces.sample(space)  # doctest: +ELLIPSIS
{'optimizer': ...}
```

**class** matchzoo.engine.hyper_spaces.**uniform**(*low*, *high*)
Bases: *matchzoo.engine.hyper_spaces.HyperoptProxy*

hyperopt.hp.uniform() proxy.

### matchzoo.engine.param module

Parameter class.

**class** matchzoo.engine.param.**Param**(*name*, *value=None*, *hyper_space=None*, *validator=None*, *desc=None*)
Bases: object

Parameter class.

Basic usages with a name and value:

```
>>> param = Param('my_param', 10)
>>> param.name
'my_param'
>>> param.value
10
```

Use with a validator to make sure the parameter always keeps a valid value.

```
>>> param = Param(
...     name='my_param',
...     value=5,
...     validator=lambda x: 0 < x < 20
... )
>>> param.validator  # doctest: +ELLIPSIS
<function <lambda> at 0x...>
>>> param.value
5
>>> param.value = 10
```

(continues on next page)

(continued from previous page)

```
>>> param.value
10
>>> param.value = -1
Traceback (most recent call last):
    ...
ValueError: Validator not satisfied.
The validator's definition is as follows:
validator=lambda x: 0 < x < 20
```

Use with a hyper space. Setting up a hyper space for a parameter makes the parameter tunable in a `matchzoo.engine.Tuner`.

```
>>> from matchzoo.engine.hyper_spaces import quniform
>>> param = Param(
...     name='positive_num',
...     value=1,
...     hyper_space=quniform(low=1, high=5)
... )
>>> param.hyper_space  # doctest: +ELLIPSIS
<matchzoo.engine.hyper_spaces.quniform object at ...>
>>> from hyperopt.pyll.stochastic import sample
>>> hyperopt_space = param.hyper_space.convert(param.name)
>>> samples = [sample(hyperopt_space) for _ in range(64)]
>>> set(samples) == {1, 2, 3, 4, 5}
True
```

The boolean value of a *Param* instance is only *True* when the value is not *None*. This is because some default falsy values like zero or an empty list are valid parameter values. In other words, the boolean value means to be "if the parameter value is filled".

```
>>> param = Param('dropout')
>>> if param:
...     print('OK')
>>> param = Param('dropout', 0)
>>> if param:
...     print('OK')
OK
```

A *_pre_assignment_hook* is initialized as a data type convertor if the value is set as a number to keep data type consistency of the parameter. This conversion supports python built-in numbers, *numpy* numbers, and any number that inherits `numbers.Number`.

```
>>> param = Param('float_param', 0.5)
>>> param.value = 10
>>> param.value
10.0
>>> type(param.value)
<class 'float'>
```

**desc**
Parameter description.

**Type** return

**Return type** `str`

**hyper_space**
Hyper space of the parameter.

**Type** return

**Return type** Union[Apply, *HyperoptProxy*]

**name**
Name of the parameter.

**Type** return

**Return type** str

**reset**()
Set the parameter's value to *None*, which means "not set".

This method bypasses validator.

### Example

```
>>> import matchzoo as mz
>>> param = mz.Param(
...     name='str', validator=lambda x: isinstance(x, str))
>>> param.value = 'hello'
>>> param.value = None
Traceback (most recent call last):
    ...
ValueError: Validator not satisfied.
The validator's definition is as follows:
name='str', validator=lambda x: isinstance(x, str))
>>> param.reset()
>>> param.value is None
True
```

**set_default**(*val*, *verbose=1*)
Set default value, has no effect if already has a value.

**Parameters**

- **val** – Default value to set.
- **verbose** – Verbosity.

**validator**
Validator of the parameter.

**Type** return

**Return type** Callable[[Any], bool]

**value**
Value of the parameter.

**Type** return

**Return type** Any

## matchzoo.engine.param_table module

Parameters table class.

**class** matchzoo.engine.param_table.**ParamTable**

Bases: object

Parameter table class.

### Example

```
>>> params = ParamTable()
>>> params.add(Param('ham', 'Parma Ham'))
>>> params.add(Param('egg', 'Over Easy'))
>>> params['ham']
'Parma Ham'
>>> params['egg']
'Over Easy'
>>> print(params)
ham                             Parma Ham
egg                             Over Easy
>>> params.add(Param('egg', 'Sunny side Up'))
Traceback (most recent call last):
    ...
ValueError: Parameter named egg already exists.
To re-assign parameter egg value, use `params["egg"] = value` instead.
```

**add**(*param*)

**Parameters** **param** (*Param*) – parameter to add.

**completed**()

**Return type** bool

**Returns** *True* if all params are filled, *False* otherwise.

### Example

```
>>> import matchzoo
>>> model = matchzoo.models.Naive()
>>> model.params.completed()
False
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.params.completed()
True
```

**get**(*key*)

**Return type** *Param*

**Returns** The parameter in the table named *key*.

**hyper_space**

Hyper space of the table, a valid *hyperopt* graph.

**Type** return

**Return type** dict

**keys**()

**Return type** KeysView

**Returns** Parameter table keys.

**set**(*key*, *param*)
Set *key* to parameter *param*.

**to_frame**()
Convert the parameter table into a pandas data frame.

**Return type** DataFrame

**Returns** A *pandas.DataFrame*.

### Example

```
>>> import matchzoo as mz
>>> table = mz.ParamTable()
>>> table.add(mz.Param(name='x', value=10, desc='my x'))
>>> table.add(mz.Param(name='y', value=20, desc='my y'))
>>> table.to_frame()
  Name Description  Value Hyper-Space
0    x        my x     10        None
1    y        my y     20        None
```

**update**(*other*)
Update *self*.

Update *self* with the key/value pairs from other, overwriting existing keys. Notice that this does not add new keys to *self*.

This method is usually used by models to obtain useful information from a preprocessor's context.

**Parameters** **other** (dict) – The dictionary used update.

### Example

```
>>> import matchzoo as mz
>>> model = mz.models.DenseBaseline()
>>> model.params['input_shapes'] is None
True
>>> prpr = model.get_default_preprocessor()
>>> _ = prpr.fit(mz.datasets.toy.load_data(), verbose=0)
>>> model.params.update(prpr.context)
>>> model.params['input_shapes']
[(30,), (30,)]
```

## matchzoo.engine.parse_metric module

matchzoo.engine.parse_metric.**parse_metric**(*metric*, *task=None*)
Parse input metric in any form into a BaseMetric instance.

**Parameters**

- **metric** (Union[str, Type[*BaseMetric*], *BaseMetric*]) – Input metric in any form.
- **task** (Optional[*BaseTask*]) – Task type for determining specific metric.

**Return type** Union[*BaseMetric*, str]

**Returns** A BaseMetric instance

**Examples::**

```
>>> from matchzoo import metrics
>>> from matchzoo.engine.parse_metric import parse_metric
```

**Use *str* as keras native metrics:**

```
>>> parse_metric('mse')
'mse'
```

**Use *str* as MatchZoo metrics:**

```
>>> mz_metric = parse_metric('map')
>>> type(mz_metric)
<class 'matchzoo.metrics.mean_average_precision.MeanAveragePrecision'>
```

**Use `matchzoo.engine.BaseMetric` subclasses as MatchZoo metrics:**

```
>>> type(parse_metric(metrics.AveragePrecision))
<class 'matchzoo.metrics.average_precision.AveragePrecision'>
```

**Use `matchzoo.engine.BaseMetric` instances as MatchZoo metrics:**

```
>>> type(parse_metric(metrics.AveragePrecision()))
<class 'matchzoo.metrics.average_precision.AveragePrecision'>
```

### Module contents

### matchzoo.layers package

### Submodules

### matchzoo.layers.dynamic_pooling_layer module

An implementation of Dynamic Pooling Layer.

**class** matchzoo.layers.dynamic_pooling_layer.**DynamicPoolingLayer**(*psize1*, *psize2*, ***kwargs*)

Bases: keras.engine.base_layer.Layer

Layer that computes dynamic pooling of one tensor.

**Parameters**

- **psize1** (int) – pooling size of dimension 1
- **psize2** (int) – pooling size of dimension 2
- **kwargs** – Standard layer keyword arguments.

### Examples

```
>>> import matchzoo as mz
>>> layer = mz.layers.DynamicPoolingLayer(3, 2)
>>> num_batch, left_len, right_len, num_dim = 5, 3, 2, 10
>>> layer.build([[num_batch, left_len, right_len, num_dim],
...              [num_batch, left_len, right_len, 3]])
```

**build**(*input_shape*)
Build the layer.

**Parameters** **input_shape** (List[int]) – the shapes of the input tensors, for DynamicPoolingLayer we need tow input tensors.

**call**(*inputs*, ***kwargs*)
The computation logic of DynamicPoolingLayer.

**Parameters** **inputs** (list) – two input tensors.

**Return type** Any

**compute_output_shape**(*input_shape*)
Calculate the layer output shape.

**Parameters** **input_shape** (list) – the shapes of the input tensors, for DynamicPoolingLayer we need tow input tensors.

**Return type** tuple

**get_config**()
Get the config dict of DynamicPoolingLayer.

**Return type** dict

## matchzoo.layers.matching_layer module

An implementation of Matching Layer.

**class** matchzoo.layers.matching_layer.**MatchingLayer**(*normalize=False*, *matching_type='dot'*, ***kwargs*)

Bases: keras.engine.base_layer.Layer

Layer that computes a matching matrix between samples in two tensors.

**Parameters**

- **normalize** (bool) – Whether to L2-normalize samples along the dot product axis before taking the dot product. If set to True, then the output of the dot product is the cosine proximity between the two samples.
- **matching_type** (str) – the similarity function for matching
- **kwargs** – Standard layer keyword arguments.

### Examples

```
>>> import matchzoo as mz
>>> layer = mz.layers.MatchingLayer(matching_type='dot',
...                                 normalize=True)
```

(continues on next page)

(continued from previous page)

```
>>> num_batch, left_len, right_len, num_dim = 5, 3, 2, 10
>>> layer.build([[num_batch, left_len, num_dim],
...              [num_batch, right_len, num_dim]])
```

**build**(*input_shape*)
Build the layer.

**Parameters** **input_shape** (list) – the shapes of the input tensors, for MatchingLayer we need tow input tensors.

**call**(*inputs*, ***kwargs*)
The computation logic of MatchingLayer.

**Parameters** **inputs** (list) – two input tensors.

**Return type** Any

**compute_output_shape**(*input_shape*)
Calculate the layer output shape.

**Parameters** **input_shape** (list) – the shapes of the input tensors, for MatchingLayer we need tow input tensors.

**Return type** tuple

**get_config**()
Get the config dict of MatchingLayer.

**Return type** dict

### Module contents

## matchzoo.losses package

### Submodules

### matchzoo.losses.rank_cross_entropy_loss module

The rank cross entropy loss.

**class** matchzoo.losses.rank_cross_entropy_loss.**RankCrossEntropyLoss**(*num_neg=1*)
Bases: object

Rank cross entropy loss.

#### Examples

```
>>> from keras import backend as K
>>> softmax = lambda x: np.exp(x)/np.sum(np.exp(x), axis=0)
>>> x_pred = K.variable(np.array([[1.0], [1.2], [0.8]]))
>>> x_true = K.variable(np.array([[1], [0], [0]]))
>>> expect = -np.log(softmax(np.array([[1.0], [1.2], [0.8]])))
>>> loss = K.eval(RankCrossEntropyLoss(num_neg=2)(x_true, x_pred))
>>> np.isclose(loss, expect[0]).all()
True
```

**num_neg**
*num_neg* getter.

### matchzoo.losses.rank_hinge_loss module

The rank hinge loss.

**class** matchzoo.losses.rank_hinge_loss.**RankHingeLoss**(*num_neg=1*, *margin=1.0*)
Bases: object

Rank hinge loss.

#### Examples

```
>>> from keras import backend as K
>>> x_pred = K.variable(np.array([[1.0], [1.2], [0.8], [1.4]]))
>>> x_true = K.variable(np.array([[1], [0], [1], [0]]))
>>> expect = ((1.0 + 1.2 - 1.0) + (1.0 + 1.4 - 0.8)) / 2
>>> expect
1.4
>>> loss = K.eval(RankHingeLoss(num_neg=1, margin=1.0)(x_true, x_pred))
>>> np.isclose(loss, expect)
True
```

**margin**
*margin* getter.

**num_neg**
*num_neg* getter.

### Module contents

### matchzoo.metrics package

### Submodules

### matchzoo.metrics.average_precision module

Average precision metric for ranking.

**class** matchzoo.metrics.average_precision.**AveragePrecision**(*threshold=0.0*)
Bases: *matchzoo.engine.base_metric.BaseMetric*

Average precision metric.

**ALIAS = ['average_precision', 'ap']**

### matchzoo.metrics.discounted_cumulative_gain module

Discounted cumulative gain metric for ranking.

**class** matchzoo.metrics.discounted_cumulative_gain.**DiscountedCumulativeGain**(*k=1*, *threshold=0.0*)

Bases: *matchzoo.engine.base_metric.BaseMetric*

Disconunted cumulative gain metric.

**ALIAS = ['discounted_cumulative_gain', 'dcg']**

### matchzoo.metrics.mean_average_precision module

Mean average precision metric for ranking.

**class** matchzoo.metrics.mean_average_precision.**MeanAveragePrecision**(*threshold=0.0*)

Bases: *matchzoo.engine.base_metric.BaseMetric*

Mean average precision metric.

**ALIAS = ['mean_average_precision', 'map']**

### matchzoo.metrics.mean_reciprocal_rank module

Mean reciprocal ranking metric.

**class** matchzoo.metrics.mean_reciprocal_rank.**MeanReciprocalRank**(*threshold=0.0*)

Bases: *matchzoo.engine.base_metric.BaseMetric*

Mean reciprocal rank metric.

**ALIAS = ['mean_reciprocal_rank', 'mrr']**

### matchzoo.metrics.normalized_discounted_cumulative_gain module

Normalized discounted cumulative gain metric for ranking.

**class** matchzoo.metrics.normalized_discounted_cumulative_gain.**NormalizedDiscountedCumulative**

Bases: *matchzoo.engine.base_metric.BaseMetric*

Normalized discounted cumulative gain metric.

**ALIAS = ['normalized_discounted_cumulative_gain', 'ndcg']**

### matchzoo.metrics.precision module

Precision for ranking.

**class** matchzoo.metrics.precision.**Precision**(*k=1*, *threshold=0.0*)

Bases: *matchzoo.engine.base_metric.BaseMetric*

Precision metric.

**ALIAS = 'precision'**

### Module contents

matchzoo.metrics.**list_available**()

**Return type** list

## matchzoo.models package

### Submodules

### matchzoo.models.anmm module

An implementation of aNMM Model.

**class** matchzoo.models.anmm.**ANMM**(*params=None*, *backend=None*)

Bases: *matchzoo.engine.base_model.BaseModel*

ANMM Model.

#### Examples

```
>>> model = ANMM()
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
```

**build**()

Build model structure.

aNMM model based on bin weighting and query term attentions

**classmethod get_default_params**()

**Return type** *ParamTable*

**Returns** model default parameters.

### matchzoo.models.arci module

An implementation of ArcI Model.

**class** matchzoo.models.arci.**ArcI**(*params=None*, *backend=None*)

Bases: *matchzoo.engine.base_model.BaseModel*

ArcI Model.

#### Examples

```
>>> model = ArcI()
>>> model.params['num_blocks'] = 1
>>> model.params['left_filters'] = [32]
>>> model.params['right_filters'] = [32]
>>> model.params['left_kernel_sizes'] = [3]
>>> model.params['right_kernel_sizes'] = [3]
>>> model.params['left_pool_sizes'] = [2]
```

(continues on next page)

(continued from previous page)

```
>>> model.params['right_pool_sizes'] = [4]
>>> model.params['conv_activation_func'] = 'relu'
>>> model.params['mlp_num_layers'] = 1
>>> model.params['mlp_num_units'] = 64
>>> model.params['mlp_num_fan_out'] = 32
>>> model.params['mlp_activation_func'] = 'relu'
>>> model.params['dropout_rate'] = 0.5
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
```

**build**()
Build model structure.

ArcI use Siamese arthitecture.

**classmethod get_default_params**()

**Return type** *ParamTable*

**Returns** model default parameters.

### matchzoo.models.arcii module

An implementation of ArcII Model.

**class** matchzoo.models.arcii.**ArcII**(*params=None*, *backend=None*)
Bases: *matchzoo.engine.base_model.BaseModel*

ArcII Model.

#### Examples

```
>>> model = ArcII()
>>> model.params['embedding_output_dim'] = 300
>>> model.params['num_blocks'] = 2
>>> model.params['kernel_1d_count'] = 32
>>> model.params['kernel_1d_size'] = 3
>>> model.params['kernel_2d_count'] = [16, 32]
>>> model.params['kernel_2d_size'] = [[3, 3], [3, 3]]
>>> model.params['pool_2d_size'] = [[2, 2], [2, 2]]
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
```

**build**()
Build model structure.

ArcII has the desirable property of letting two sentences meet before their own high-level representations mature.

**classmethod get_default_params**()

**Return type** *ParamTable*

**Returns** model default parameters.

### matchzoo.models.cdssm module

An implementation of CDSSM (CLSM) model.

**class** matchzoo.models.cdssm.**CDSSM**(*params=None*, *backend=None*)
Bases: *matchzoo.engine.base_model.BaseModel*

CDSSM Model implementation.

Learning Semantic Representations Using Convolutional Neural Networks for Web Search. (2014a) A Latent Semantic Model with Convolutional-Pooling Structure for Information Retrieval. (2014b)

#### Examples

```
>>> model = CDSSM()
>>> model.params['optimizer'] = 'adam'
>>> model.params['filters'] =  32
>>> model.params['kernel_size'] = 3
>>> model.params['conv_activation_func'] = 'relu'
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
```

**build**()
Build model structure.

CDSSM use Siamese architecture.

**classmethod get_default_params**()

**Return type** *ParamTable*

**Returns** model default parameters.

**classmethod get_default_preprocessor**()

**Returns** Default preprocessor.

**guess_and_fill_missing_params**(*verbose=1*)
Guess and fill missing parameters in `params`.

Use this method to automatically fill-in hyper parameters. This involves some guessing so the parameter it fills could be wrong. For example, the default task is *Ranking*, and if we do not set it to *Classification* manually for data packs prepared for classification, then the shape of the model output and the data will mismatch.

**Parameters** **verbose** (`int`) – Verbosity.

### matchzoo.models.conv_knrm module

ConvKNRM model.

**class** matchzoo.models.conv_knrm.**ConvKNRM**(*params=None*, *backend=None*)
Bases: *matchzoo.models.knrm.KNRM*

ConvKNRM model.

#### Examples

```
>>> model = ConvKNRM()
>>> model.params['embedding_input_dim'] = 10000
>>> model.params['embedding_output_dim'] = 300
>>> model.params['embedding_trainable'] = True
>>> model.params['filters'] = 128
>>> model.params['conv_activation_func'] = 'tanh'
>>> model.params['max_ngram'] = 3
>>> model.params['use_crossmatch'] = True
>>> model.params['kernel_num'] = 11
>>> model.params['sigma'] = 0.1
>>> model.params['exact_sigma'] = 0.001
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
```

**build**()
Build model.

**get_default_params**()
Get default parameters.

### matchzoo.models.dense_baseline module

A simple densely connected baseline model.

**class** matchzoo.models.dense_baseline.**DenseBaseline**(*params=None*, *backend=None*)
Bases: *matchzoo.engine.base_model.BaseModel*

A simple densely connected baseline model.

#### Examples

```
>>> model = DenseBaseline()
>>> model.params['mlp_num_layers'] = 2
>>> model.params['mlp_num_units'] = 300
>>> model.params['mlp_num_fan_out'] = 128
>>> model.params['mlp_activation_func'] = 'relu'
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
>>> model.compile()
```

**build**()
Model structure.

**classmethod get_default_params**()

**Return type** *ParamTable*

**Returns** model default parameters.

### matchzoo.models.drmm module

An implementation of DRMM Model.

**class** matchzoo.models.drmm.**DRMM**(*params=None*, *backend=None*)

Bases: *matchzoo.engine.base_model.BaseModel*

DRMM Model.

### Examples

```
>>> model = DRMM()
>>> model.params['mlp_num_layers'] = 1
>>> model.params['mlp_num_units'] = 5
>>> model.params['mlp_num_fan_out'] = 1
>>> model.params['mlp_activation_func'] = 'tanh'
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
>>> model.compile()
```

**classmethod attention_layer**(*attention_input*, *attention_mask=None*)

Performs attention on the input.

**Parameters**

- **attention_input** (Any) – The input tensor for attention layer.
- **attention_mask** (Optional[Any]) – A tensor to mask the invalid values.

**Return type** Layer

**Returns** The masked output tensor.

**build**()

Build model structure.

**classmethod get_default_params**()

**Return type** *ParamTable*

**Returns** model default parameters.

## matchzoo.models.drmmtks module

An implementation of DRMMTKS Model.

**class** matchzoo.models.drmmtks.**DRMMTKS**(*params=None*, *backend=None*)

Bases: *matchzoo.engine.base_model.BaseModel*

DRMMTKS Model.

### Examples

```
>>> model = DRMMTKS()
>>> model.params['embedding_input_dim'] = 10000
>>> model.params['embedding_output_dim'] = 100
>>> model.params['top_k'] = 20
>>> model.params['mlp_num_layers'] = 1
>>> model.params['mlp_num_units'] = 5
>>> model.params['mlp_num_fan_out'] = 1
>>> model.params['mlp_activation_func'] = 'tanh'
```

(continues on next page)

(continued from previous page)

```
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
```

**classmethod attention_layer**(*attention_input*, *attention_mask=None*)

Performs attention on the input.

**Parameters**

- **attention_input** (Any) – The input tensor for attention layer.
- **attention_mask** (Optional[Any]) – A tensor to mask the invalid values.

**Return type** Layer

**Returns** The masked output tensor.

**build**()

Build model structure.

**classmethod get_default_params**()

**Return type** *ParamTable*

**Returns** model default parameters.

### matchzoo.models.dssm module

An implementation of DSSM, Deep Structured Semantic Model.

**class** matchzoo.models.dssm.**DSSM**(*params=None*, *backend=None*)

Bases: *matchzoo.engine.base_model.BaseModel*

Deep structured semantic model.

#### Examples

```
>>> model = DSSM()
>>> model.params['mlp_num_layers'] = 3
>>> model.params['mlp_num_units'] = 300
>>> model.params['mlp_num_fan_out'] = 128
>>> model.params['mlp_activation_func'] = 'relu'
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
```

**build**()

Build model structure.

DSSM use Siamese arthitecture.

**classmethod get_default_params**()

**Return type** *ParamTable*

**Returns** model default parameters.

**classmethod get_default_preprocessor**()

**Returns** Default preprocessor.

### matchzoo.models.duet module

DUET Model.

**class** matchzoo.models.duet.**DUET**(*params=None*, *backend=None*)
Bases: *matchzoo.engine.base_model.BaseModel*

DUET Model.

#### Examples

```
>>> model = DUET()
>>> model.params['embedding_input_dim'] = 1000
>>> model.params['embedding_output_dim'] = 300
>>> model.params['lm_filters'] = 32
>>> model.params['lm_hidden_sizes'] = [64, 32]
>>> model.params['dropout_rate'] = 0.5
>>> model.params['dm_filters'] = 32
>>> model.params['dm_kernel_size'] = 3
>>> model.params['dm_d_mpool'] = 4
>>> model.params['dm_hidden_sizes'] = [64, 32]
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
```

**build**()
Build model.

**classmethod get_default_params**()
Get default parameters.

### matchzoo.models.knrm module

KNRM model.

**class** matchzoo.models.knrm.**KNRM**(*params=None*, *backend=None*)
Bases: *matchzoo.engine.base_model.BaseModel*

KNRM model.

#### Examples

```
>>> model = KNRM()
>>> model.params['embedding_input_dim'] =  10000
>>> model.params['embedding_output_dim'] =  10
>>> model.params['embedding_trainable'] = True
>>> model.params['kernel_num'] = 11
>>> model.params['sigma'] = 0.1
>>> model.params['exact_sigma'] = 0.001
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
```

**build**()
Build model.

**classmethod get_default_params**()
Get default parameters.

### matchzoo.models.match_pyramid module

An implementation of MatchPyramid Model.

**class** matchzoo.models.match_pyramid.**MatchPyramid**(*params=None*, *backend=None*)
Bases: *matchzoo.engine.base_model.BaseModel*

MatchPyramid Model.

#### Examples

```
>>> model = MatchPyramid()
>>> model.params['embedding_output_dim'] = 300
>>> model.params['num_blocks'] = 2
>>> model.params['kernel_count'] = [16, 32]
>>> model.params['kernel_size'] = [[3, 3], [3, 3]]
>>> model.params['dpool_size'] = [3, 10]
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
```

**build**()
Build model structure.

MatchPyramid text matching as image recognition.

**classmethod get_default_params**()

**Return type** *ParamTable*

**Returns** model default parameters.

### matchzoo.models.mvlstm module

An implementation of MVLSTM Model.

**class** matchzoo.models.mvlstm.**MVLSTM**(*params=None*, *backend=None*)
Bases: *matchzoo.engine.base_model.BaseModel*

MVLSTM Model.

#### Examples

```
>>> model = MVLSTM()
>>> model.params['lstm_units'] = 32
>>> model.params['top_k'] = 50
>>> model.params['mlp_num_layers'] = 2
>>> model.params['mlp_num_units'] = 20
>>> model.params['mlp_num_fan_out'] = 10
>>> model.params['mlp_activation_func'] = 'relu'
>>> model.params['dropout_rate'] = 0.5
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
```

**build**()
Build model structure.

**classmethod get_default_params**()

**Return type** *ParamTable*

**Returns** model default parameters.

### matchzoo.models.naive module

Naive model with a simplest structure for testing purposes.

**class** matchzoo.models.naive.**Naive**(*params=None*, *backend=None*)
Bases: *matchzoo.engine.base_model.BaseModel*

Naive model with a simplest structure for testing purposes.

Bare minimum functioning model. The best choice to get things rolling. The worst choice to fit and evaluate performance.

**build**()
Build.

**classmethod get_default_params**()
Default parameters.

### matchzoo.models.parameter_readme_generator module

matchzoo/models/README.md generater.

### Module contents

matchzoo.models.**list_available**()

**Return type** list

### matchzoo.preprocessors package

### Subpackages

### matchzoo.preprocessors.units package

### Submodules

### matchzoo.preprocessors.units.digit_removal module

**class** matchzoo.preprocessors.units.digit_removal.**DigitRemoval**
Bases: *matchzoo.preprocessors.units.unit.Unit*

Process unit to remove digits.

**transform**(*input_*)
Remove digits from list of tokens.

**Parameters** **input** – list of tokens to be filtered.

**Return tokens** tokens of tokens without digits.

**Return type** list

### matchzoo.preprocessors.units.fixed_length module

**class** matchzoo.preprocessors.units.fixed_length.**FixedLength**(*text_length*, *pad_value=0*, *pad_mode='pre'*, *truncate_mode='pre'*)

Bases: *matchzoo.preprocessors.units.unit.Unit*

FixedLengthUnit Class.

Process unit to get the fixed length text.

#### Examples

```
>>> from matchzoo.preprocessors.units import FixedLength
>>> fixedlen = FixedLength(3)
>>> fixedlen.transform(list(range(1, 6))) == [3, 4, 5]
True
>>> fixedlen.transform(list(range(1, 3))) == [0, 1, 2]
True
```

**transform**(*input_*)

Transform list of tokenized tokens into the fixed length text.

**Parameters** **input** – list of tokenized tokens.

**Return tokens** list of tokenized tokens in fixed length.

**Return type** list

### matchzoo.preprocessors.units.frequency_filter module

**class** matchzoo.preprocessors.units.frequency_filter.**FrequencyFilter**(*low=0*, *high=inf*, *mode='df'*)

Bases: *matchzoo.preprocessors.units.stateful_unit.StatefulUnit*

Frequency filter unit.

**Parameters**

- **low** (float) – Lower bound, inclusive.
- **high** (float) – Upper bound, exclusive.
- **mode** (str) – One of *tf* (term frequency), *df* (document frequency), and *idf* (inverse document frequency).

**Examples::**

```
>>> import matchzoo as mz
```

**To filter based on term frequency (tf):**

```
>>> tf_filter = mz.preprocessors.units.FrequencyFilter(
...     low=2, mode='tf')
>>> tf_filter.fit([['A', 'B', 'B'], ['C', 'C', 'C']])
>>> tf_filter.transform(['A', 'B', 'C'])
['B', 'C']
```

**To filter based on document frequency (df):**

```
>>> tf_filter = mz.preprocessors.units.FrequencyFilter(
...     low=2, mode='df')
>>> tf_filter.fit([['A', 'B'], ['B', 'C']])
>>> tf_filter.transform(['A', 'B', 'C'])
['B']
```

**To filter based on inverse document frequency (idf):**

```
>>> idf_filter = mz.preprocessors.units.FrequencyFilter(
...     low=1.2, mode='idf')
>>> idf_filter.fit([['A', 'B'], ['B', 'C', 'D']])
>>> idf_filter.transform(['A', 'B', 'C'])
['A', 'C']
```

**fit**(*list_of_tokens*)
Fit *list_of_tokens* by calculating *mode* states.

**transform**(*input_*)
Transform a list of tokens by filtering out unwanted words.

**Return type** list

### matchzoo.preprocessors.units.lemmatization module

**class** matchzoo.preprocessors.units.lemmatization.**Lemmatization**
Bases: *matchzoo.preprocessors.units.unit.Unit*

Process unit for token lemmatization.

**transform**(*input_*)
Lemmatization a sequence of tokens.

**Parameters** **input** – list of tokens to be lemmatized.

**Return tokens** list of lemmatizd tokens.

**Return type** list

### matchzoo.preprocessors.units.lowercase module

**class** matchzoo.preprocessors.units.lowercase.**Lowercase**
Bases: *matchzoo.preprocessors.units.unit.Unit*

Process unit for text lower case.

**transform**(*input_*)
Convert list of tokens to lower case.

**Parameters** **input** – list of tokens.

**Return tokens** lower-cased list of tokens.

**Return type** `list`

### matchzoo.preprocessors.units.matching_histogram module

**class** matchzoo.preprocessors.units.matching_histogram.**MatchingHistogram**(*bin_size=30*, *embedding_matrix=None*, *normalize=True*, *mode='LCH'*)

Bases: *matchzoo.preprocessors.units.unit.Unit*

MatchingHistogramUnit Class.

**Parameters**

- **bin_size** (`int`) – The number of bins of the matching histogram.
- **embedding_matrix** – The word embedding matrix applied to calculate the matching histogram.
- **normalize** – Boolean, normalize the embedding or not.
- **mode** (`str`) – The type of the historgram, it should be one of 'CH', 'NG', or 'LCH'.

#### Examples

```
>>> embedding_matrix = np.array([[1.0, -1.0], [1.0, 2.0], [1.0, 3.0]])
>>> text_left = [0, 1]
>>> text_right = [1, 2]
>>> histogram = MatchingHistogram(3, embedding_matrix, True, 'CH')
>>> histogram.transform([text_left, text_right])
[[3.0, 1.0, 1.0], [1.0, 2.0, 2.0]]
```

**transform**(*input_*)

Transform the input text.

**Return type** `list`

### matchzoo.preprocessors.units.ngram_letter module

**class** matchzoo.preprocessors.units.ngram_letter.**NgramLetter**(*ngram=3*, *reduce_dim=True*)

Bases: *matchzoo.preprocessors.units.unit.Unit*

Process unit for n-letter generation.

Triletter is used in `DSSMModel`. This processor is expected to execute before *Vocab* has been created.

#### Examples

```
>>> triletter = NgramLetter()
>>> rv = triletter.transform(['hello', 'word'])
>>> len(rv)
9
>>> rv
['#he', 'hel', 'ell', 'llo', 'lo#', '#wo', 'wor', 'ord', 'rd#']
>>> triletter = NgramLetter(reduce_dim=False)
>>> rv = triletter.transform(['hello', 'word'])
>>> len(rv)
2
>>> rv
[['#he', 'hel', 'ell', 'llo', 'lo#'], ['#wo', 'wor', 'ord', 'rd#']]
```

**transform**(*input_*)

Transform token into tri-letter.

For example, *word* should be represented as *#wo*, *wor*, *ord* and *rd#*.

**Parameters** **input** – list of tokens to be transformed.

**Return n_letters** generated n_letters.

**Return type** list

### matchzoo.preprocessors.units.punc_removal module

**class** matchzoo.preprocessors.units.punc_removal.**PuncRemoval**

Bases: *matchzoo.preprocessors.units.unit.Unit*

Process unit for remove punctuations.

**transform**(*input_*)

Remove punctuations from list of tokens.

**Parameters** **input** – list of toekns.

**Return rv** tokens without punctuation.

**Return type** list

### matchzoo.preprocessors.units.stateful_unit module

**class** matchzoo.preprocessors.units.stateful_unit.**StatefulUnit**

Bases: *matchzoo.preprocessors.units.unit.Unit*

Unit with inner state.

Usually need to be fit before transforming. All information gathered in the fit phrase will be stored into its *context*.

**context**

Get current context. Same as *unit.state*.

**fit**(*input_*)

Abstract base method, need to be implemented in subclass.

**state**
Get current context. Same as *unit.context*.

Deprecated since v2.2.0, and will be removed in the future. Used *unit.context* instead.

### matchzoo.preprocessors.units.stemming module

**class** matchzoo.preprocessors.units.stemming.**Stemming**(*stemmer='porter'*)
Bases: *matchzoo.preprocessors.units.unit.Unit*

Process unit for token stemming.

**Parameters** **stemmer** – stemmer to use, *porter* or *lancaster*.

**transform**(*input_*)
Reducing inflected words to their word stem, base or root form.

**Parameters** **input** – list of string to be stemmed.

**Return type** list

### matchzoo.preprocessors.units.stop_removal module

**class** matchzoo.preprocessors.units.stop_removal.**StopRemoval**(*lang='english'*)
Bases: *matchzoo.preprocessors.units.unit.Unit*

Process unit to remove stop words.

#### Example

```
>>> unit = StopRemoval()
>>> unit.transform(['a', 'the', 'test'])
['test']
>>> type(unit.stopwords)
<class 'list'>
```

**stopwords**
Get stopwords based on language.

**Params lang** language code.

**Return type** list

**Returns** list of stop words.

**transform**(*input_*)
Remove stopwords from list of tokenized tokens.

**Parameters**

- **input** – list of tokenized tokens.
- **lang** – language code for stopwords.

**Return tokens** list of tokenized tokens without stopwords.

**Return type** list

### matchzoo.preprocessors.units.tokenize module

**class** matchzoo.preprocessors.units.tokenize.**Tokenize**
Bases: *matchzoo.preprocessors.units.unit.Unit*

Process unit for text tokenization.

**transform**(*input_*)
Process input data from raw terms to list of tokens.

**Parameters** **input** – raw textual input.

**Return tokens** tokenized tokens as a list.

**Return type** list

### matchzoo.preprocessors.units.unit module

**class** matchzoo.preprocessors.units.unit.**Unit**
Bases: object

Process unit do not persive state (i.e. do not need fit).

**transform**(*input_*)
Abstract base method, need to be implemented in subclass.

### matchzoo.preprocessors.units.vocabulary module

**class** matchzoo.preprocessors.units.vocabulary.**Vocabulary**(*pad_value='<PAD>'*, *oov_value='<OOV>'*)
Bases: *matchzoo.preprocessors.units.stateful_unit.StatefulUnit*

Vocabulary class.

**Parameters**

- **pad_value** (str) – The string value for the padding position.
- **oov_value** (str) – The string value for the out-of-vocabulary terms.

#### Examples

```
>>> vocab = Vocabulary(pad_value='[PAD]', oov_value='[OOV]')
>>> vocab.fit(['A', 'B', 'C', 'D', 'E'])
>>> term_index = vocab.state['term_index']
>>> term_index  # doctest: +SKIP
{'[PAD]': 0, '[OOV]': 1, 'D': 2, 'A': 3, 'B': 4, 'C': 5, 'E': 6}
>>> index_term = vocab.state['index_term']
>>> index_term  # doctest: +SKIP
{0: '[PAD]', 1: '[OOV]', 2: 'D', 3: 'A', 4: 'B', 5: 'C', 6: 'E'}
```

```
>>> term_index['out-of-vocabulary-term']
1
>>> index_term[0]
'[PAD]'
>>> index_term[42]
```

(continues on next page)

(continued from previous page)

```
Traceback (most recent call last):
    ...
KeyError: 42
>>> a_index = term_index['A']
>>> c_index = term_index['C']
>>> vocab.transform(['C', 'A', 'C']) == [c_index, a_index, c_index]
True
>>> vocab.transform(['C', 'A', '[OOV]']) == [c_index, a_index, 1]
True
>>> indices = vocab.transform(list('ABCDDZZZ'))
>>> ' '.join(vocab.state['index_term'][i] for i in indices)
'A B C D D [OOV] [OOV] [OOV]'
```

**class TermIndex**
Bases: dict

Map term to index.

**fit**(*tokens*)
Build a *TermIndex* and a IndexTerm.

**transform**(*input_*)
Transform a list of tokens to corresponding indices.

**Return type** list

### matchzoo.preprocessors.units.word_hashing module

**class** matchzoo.preprocessors.units.word_hashing.**WordHashing**(*term_index*)
Bases: *matchzoo.preprocessors.units.unit.Unit*

Word-hashing layer for DSSM-based models.

The input of WordHashingUnit should be a list of word sub-letter list extracted from one document. The output of is the word-hashing representation of this document.

NgramLetterUnit and VocabularyUnit are two essential prerequisite of WordHashingUnit.

#### Examples

```
>>> letters = [['#te', 'tes','est', 'st#'], ['oov']]
>>> word_hashing = WordHashing(
...     term_index={
...      '_PAD': 0, 'OOV': 1, 'st#': 2, '#te': 3, 'est': 4, 'tes': 5
...      })
>>> hashing = word_hashing.transform(letters)
>>> hashing[0]
[0.0, 0.0, 1.0, 1.0, 1.0, 1.0]
>>> hashing[1]
[0.0, 1.0, 0.0, 0.0, 0.0, 0.0]
```

**transform**(*input_*)
Transform list of letters into word hashing layer.

**Parameters** **input** – list of *tri_letters* generated by NgramLetterUnit.

**Return type** list

**Returns** Word hashing representation of *tri-letters*.

### Module contents

matchzoo.preprocessors.units.**list_available**()

**Return type** list

### Submodules

### matchzoo.preprocessors.basic_preprocessor module

Basic Preprocessor.

**class** matchzoo.preprocessors.basic_preprocessor.**BasicPreprocessor**(*fixed_length_left=30*, *fixed_length_right=30*, *filter_mode='df'*, *filter_low_freq=2*, *filter_high_freq=inf*, *remove_stop_words=False*)

Bases: *matchzoo.engine.base_preprocessor.BasePreprocessor*

Baisc preprocessor helper.

**Parameters**

- **fixed_length_left** (int) – Integer, maximize length of left in the data_pack.
- **fixed_length_right** (int) – Integer, maximize length of right in the data_pack.
- **filter_mode** (str) – String, mode used by FrequenceFilterUnit, Can be 'df', 'cf', and 'idf'.
- **filter_low_freq** (float) – Float, lower bound value used by FrequenceFilterUnit.
- **filter_high_freq** (float) – Float, upper bound value used by FrequenceFilterUnit.
- **remove_stop_words** (bool) – Bool, use StopRemovalUnit unit or not.

#### Example

```
>>> import matchzoo as mz
>>> train_data = mz.datasets.toy.load_data('train')
>>> test_data = mz.datasets.toy.load_data('test')
>>> preprocessor = mz.preprocessors.BasicPreprocessor(
...     fixed_length_left=10,
...     fixed_length_right=20,
...     filter_mode='df',
...     filter_low_freq=2,
...     filter_high_freq=1000,
```

(continues on next page)

(continued from previous page)

```
...     remove_stop_words=True
... )
>>> preprocessor = preprocessor.fit(train_data, verbose=0)
>>> preprocessor.context['input_shapes']
[(10,), (20,)]
>>> preprocessor.context['vocab_size']
226
>>> processed_train_data = preprocessor.transform(train_data,
...                                               verbose=0)
>>> type(processed_train_data)
<class 'matchzoo.data_pack.data_pack.DataPack'>
>>> test_data_transformed = preprocessor.transform(test_data,
...                                                verbose=0)
>>> type(test_data_transformed)
<class 'matchzoo.data_pack.data_pack.DataPack'>
```

**fit**(*data_pack*, *verbose=1*)

Fit pre-processing context for transformation.

**Parameters**

- **data_pack** (*DataPack*) – data_pack to be preprocessed.
- **verbose** (int) – Verbosity.

**Returns** class:*BasicPreprocessor* instance.

**transform**(*data_pack*, *verbose=1*)

Apply transformation on data, create fixed length representation.

**Parameters**

- **data_pack** (*DataPack*) – Inputs to be preprocessed.
- **verbose** (int) – Verbosity.

**Return type** *DataPack*

**Returns** Transformed data as DataPack object.

### matchzoo.preprocessors.build_unit_from_data_pack module

Build unit from data pack.

matchzoo.preprocessors.build_unit_from_data_pack.**build_unit_from_data_pack**(*unit*, *data_pack*, *mode='both'*, *flatten=True*, *verbose=1*)

Build a StatefulUnit from a DataPack object.

**Parameters**

- **unit** (*StatefulUnit*) – StatefulUnit object to be built.
- **data_pack** (*DataPack*) – The input DataPack object.
- **mode** (str) – One of 'left', 'right', and 'both', to determine the source data for building the VocabularyUnit.

- **flatten** (bool) – Flatten the datapack or not. *True* to organize the DataPack text as a list, and *False* to organize DataPack text as a list of list.
- **verbose** (int) – Verbosity.

**Return type** *StatefulUnit*

**Returns** A built StatefulUnit object.

### matchzoo.preprocessors.build_vocab_unit module

matchzoo.preprocessors.build_vocab_unit.**build_vocab_unit**(*data_pack*, *mode='both'*, *verbose=1*)

Build a preprocessor.units.Vocabulary given *data_pack*.

The *data_pack* should be preprocessed forehand, and each item in *text_left* and *text_right* columns of the *data_pack* should be a list of tokens.

**Parameters**

- **data_pack** (*DataPack*) – The DataPack to build vocabulary upon.
- **mode** (str) – One of 'left', 'right', and 'both', to determine the source

data for building the VocabularyUnit. :type verbose: int :param verbose: Verbosity. :rtype: *Vocabulary* :return: A built vocabulary unit.

### matchzoo.preprocessors.cdssm_preprocessor module

CDSSM Preprocessor.

**class** matchzoo.preprocessors.cdssm_preprocessor.**CDSSMPreprocessor**(*fixed_length_left=10*, *fixed_length_right=40*, *with_word_hashing=True*)

Bases: *matchzoo.engine.base_preprocessor.BasePreprocessor*

CDSSM Model preprocessor.

**fit**(*data_pack*, *verbose=1*)

Fit pre-processing context for transformation.

**Parameters**

- **verbose** (int) – Verbosity.
- **data_pack** (*DataPack*) – Data_pack to be preprocessed.

**Returns** class:*CDSSMPreprocessor* instance.

**transform**(*data_pack*, *verbose=1*)

Apply transformation on data, create *letter-ngram* representation.

**Parameters**

- **data_pack** (*DataPack*) – Inputs to be preprocessed.
- **verbose** (int) – Verbosity.

**Return type** *DataPack*

**Returns** Transformed data as DataPack object.

**with_word_hashing**
*with_word_hashing* getter.

### matchzoo.preprocessors.chain_transform module

Wrapper function organizes a number of transform functions.

matchzoo.preprocessors.chain_transform.**chain_transform**(*units*)
Compose unit transformations into a single function.

**Parameters** **units** (List[*Unit*]) – List of matchzoo.StatelessUnit.

**Return type** Callable

### matchzoo.preprocessors.dssm_preprocessor module

DSSM Preprocessor.

**class** matchzoo.preprocessors.dssm_preprocessor.**DSSMPreprocessor**(*with_word_hashing=True*)
Bases: *matchzoo.engine.base_preprocessor.BasePreprocessor*

DSSM Model preprocessor.

**fit**(*data_pack*, *verbose=1*)
Fit pre-processing context for transformation.

**Parameters**

- **verbose** (int) – Verbosity.
- **data_pack** (*DataPack*) – data_pack to be preprocessed.

**Returns** class:*DSSMPreprocessor* instance.

**transform**(*data_pack*, *verbose=1*)
Apply transformation on data, create *tri-letter* representation.

**Parameters**

- **data_pack** (*DataPack*) – Inputs to be preprocessed.
- **verbose** (int) – Verbosity.

**Return type** *DataPack*

**Returns** Transformed data as DataPack object.

**with_word_hashing**
*with_word_hashing* getter.

### matchzoo.preprocessors.naive_preprocessor module

Naive Preprocessor.

**class** matchzoo.preprocessors.naive_preprocessor.**NaivePreprocessor**
Bases: *matchzoo.engine.base_preprocessor.BasePreprocessor*

Naive preprocessor.

### Example

```
>>> import matchzoo as mz
>>> train_data = mz.datasets.toy.load_data()
>>> test_data = mz.datasets.toy.load_data(stage='test')
>>> preprocessor = mz.preprocessors.NaivePreprocessor()
>>> train_data_processed = preprocessor.fit_transform(train_data,
...                                                   verbose=0)
>>> type(train_data_processed)
<class 'matchzoo.data_pack.data_pack.DataPack'>
>>> test_data_transformed = preprocessor.transform(test_data,
...                                                verbose=0)
>>> type(test_data_transformed)
<class 'matchzoo.data_pack.data_pack.DataPack'>
```

**fit**(*data_pack*, *verbose=1*)

Fit pre-processing context for transformation.

**Parameters**

- **data_pack** (*DataPack*) – data_pack to be preprocessed.
- **verbose** (int) – Verbosity.

**Returns** class:*NaivePreprocessor* instance.

**transform**(*data_pack*, *verbose=1*)

Apply transformation on data, create *tri-letter* representation.

**Parameters**

- **data_pack** (*DataPack*) – Inputs to be preprocessed.
- **verbose** (int) – Verbosity.

**Return type** *DataPack*

**Returns** Transformed data as DataPack object.

### Module contents

matchzoo.preprocessors.**list_available**()

**Return type** list

## matchzoo.tasks package

### Submodules

### matchzoo.tasks.classification module

Classification task.

**class** matchzoo.tasks.classification.**Classification**(*num_classes=2*, ***kwargs*)

Bases: *matchzoo.engine.base_task.BaseTask*

Classification task.

### Examples

```
>>> classification_task = Classification(num_classes=2)
>>> classification_task.metrics = ['precision']
>>> classification_task.num_classes
2
>>> classification_task.output_shape
(2,)
>>> classification_task.output_dtype
<class 'int'>
>>> print(classification_task)
Classification Task with 2 classes
```

**classmethod list_available_losses**()

**Return type** list

**Returns** a list of available losses.

**classmethod list_available_metrics**()

**Return type** list

**Returns** a list of available metrics.

**num_classes**
number of classes to classify.

**Type** return

**Return type** int

**output_dtype**
target data type, expect *int* as output.

**Type** return

**output_shape**
output shape of a single sample of the task.

**Type** return

**Return type** tuple

## matchzoo.tasks.ranking module

Ranking task.

**class** matchzoo.tasks.ranking.**Ranking**(*loss=None*, *metrics=None*)
Bases: *matchzoo.engine.base_task.BaseTask*

Ranking Task.

### Examples

```
>>> ranking_task = Ranking()
>>> ranking_task.metrics = ['map', 'ndcg']
>>> ranking_task.output_shape
(1,)
```

(continues on next page)

(continued from previous page)

```
>>> ranking_task.output_dtype
<class 'float'>
>>> print(ranking_task)
Ranking Task
```

**classmethod list_available_losses**()

**Return type** list

**Returns** a list of available losses.

**classmethod list_available_metrics**()

**Return type** list

**Returns** a list of available metrics.

**output_dtype**
target data type, expect *float* as output.

**Type** return

**output_shape**
output shape of a single sample of the task.

**Type** return

**Return type** tuple

### Module contents

### matchzoo.utils package

### Submodules

### matchzoo.utils.list_recursive_subclasses module

matchzoo.utils.list_recursive_subclasses.**list_recursive_concrete_subclasses**(*base*)
List all concrete subclasses of *base* recursively.

### matchzoo.utils.make_keras_optimizer_picklable module

matchzoo.utils.make_keras_optimizer_picklable.**make_keras_optimizer_picklable**()
Fix https://github.com/NTMC-Community/MatchZoo/issues/726.

This function changes how keras behaves, use with caution.

### matchzoo.utils.one_hot module

One hot vectors.

matchzoo.utils.one_hot.**one_hot**(*indices*, *num_classes*)

**Return type** ndarray

**Returns** A one-hot encoded vector.

#### matchzoo.utils.tensor_type module

Define Keras tensor type.

#### Module contents

### 1.1.2 Submodules

### 1.1.3 matchzoo.version module

Matchzoo version file.

### 1.1.4 Module contents

CHAPTER 2

# MatchZoo Model Reference

## 2.1 Naive

### 2.1.1 Model Documentation

Naive model with a simplest structure for testing purposes.

Bare minimum functioning model. The best choice to get things rolling. The worst choice to fit and evaluate performance.

### 2.1.2 Model Hyper Parameters

| | Name | Description | Default Value | Default Hyper-Space |
|---|---|---|---|---|
| 0 | model_class | Model class. Used internally for save/load. Changing this may cause unexpected behaviors. | <class 'matchzoo.models.naive.Naive'> | |
| 1 | input_shapes | Dependent on the model and data. Should be set manually. | | |
| 2 | task | Decides model output shape, loss, and metrics. | | |
| 3 | optimizer | | adam | choice in ['adam', 'adagrad', 'rmsprop'] |

## 2.2 DSSM

### 2.2.1 Model Documentation

Deep structured semantic model.

**Examples:**

```
>>> model = DSSM()
>>> model.params['mlp_num_layers'] = 3
>>> model.params['mlp_num_units'] = 300
>>> model.params['mlp_num_fan_out'] = 128
>>> model.params['mlp_activation_func'] = 'relu'
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
```

### 2.2.2 Model Hyper Parameters

| | Name | Description | Default Value | Default Hyper-Space |
|---|---|---|---|---|
| 0 | model_class | Model class. Used internally for save/load. Changing this may cause unexpected behaviors. | <class 'matchzoo.models.dssm.DSSM'> | |
| 1 | input_shapes | Dependent on the model and data. Should be set manually. | | |
| 2 | task | Decides model output shape, loss, and metrics. | | |
| 3 | optimizer | | adam | |
| 4 | with_multi_layer_perceptron | A flag of whether a multiple layer perceptron is used. Shouldn't be changed. | True | |
| 5 | mlp_num_units | Number of units in first *mlp_num_layers* layers. | 128 | quantitative uniform distribution in [8, 256), with a step size of 8 |
| 6 | mlp_num_layers | Number of layers of the multiple layer percetron. | 3 | quantitative uniform distribution in [1, 6), with a step size of 1 |
| 7 | mlp_num_fan_out | Number of units of the layer that connects the multiple layer percetron and the output. | 64 | quantitative uniform distribution in [4, 128), with a step size of 4 |
| 8 | mlp_activation_func | Activation function used in the multiple layer perceptron. | relu | |

## 2.3 CDSSM

### 2.3.1 Model Documentation

CDSSM Model implementation.

Learning Semantic Representations Using Convolutional Neural Networks for Web Search. (2014a) A Latent Semantic Model with Convolutional-Pooling Structure for Information Retrieval. (2014b)

**Examples:**

```
>>> model = CDSSM()
>>> model.params['optimizer'] = 'adam'
>>> model.params['filters'] =  32
>>> model.params['kernel_size'] = 3
>>> model.params['conv_activation_func'] = 'relu'
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
```

### 2.3.2 Model Hyper Parameters

| | Name | Description | Default Value | Default Hyper-Space |
|---|---|---|---|---|
| 0 | model_class | Model class. Used internally for save/load. Changing this may cause unexpected behaviors. | <class 'matchzoo.models.cdssm.CDSSM'> | |
| 1 | input_shapes | Dependent on the model and data. Should be set manually. | | |
| 2 | task | Decides model output shape, loss, and metrics. | | |
| 3 | optimizer | | adam | |
| 4 | with_multi_layer_perceptron | A flag of whether a multiple layer perceptron is used. Shouldn't be changed. | True | |
| 5 | mlp_num_units | Number of units in first *mlp_num_layers* layers. | 128 | quantitative uniform distribution in [8, 256), with a step size of 8 |
| 6 | mlp_num_layers | Number of layers of the multiple layer percetron. | 3 | quantitative uniform distribution in [1, 6), with a step size of 1 |
| 7 | mlp_num_fan_out | Number of units of the layer that connects the multiple layer percetron and the output. | 64 | quantitative uniform distribution in [4, 128), with a step size of 4 |
| 8 | mlp_activation_func | Activation function used in the multiple layer perceptron. | relu | |
| 9 | filters | Number of filters in the 1D convolution layer. | 32 | |
| 10 | kernel_size | Number of kernel size in the 1D convolution layer. | 3 | |
| 11 | strides | Strides in the 1D convolution layer. | 1 | |
| 12 | padding | The padding mode in the convolution layer. It should be one of *same*, *valid*, and *causal*. | same | |
| 13 | conv_activation_func | Activation function in the convolution layer. | relu | |
| 14 | w_initializer | | glorot_normal | |
| 15 | b_initializer | | zeros | |
| 16 | dropout_rate | The dropout rate. | 0.3 | |

## 2.4 DenseBaseline

### 2.4.1 Model Documentation

A simple densely connected baseline model.

**Examples:**

```
>>> model = DenseBaseline()
>>> model.params['mlp_num_layers'] = 2
>>> model.params['mlp_num_units'] = 300
>>> model.params['mlp_num_fan_out'] = 128
>>> model.params['mlp_activation_func'] = 'relu'
```

(continues on next page)

(continued from previous page)

```
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
>>> model.compile()
```

### 2.4.2 Model Hyper Parameters

| | Name | Description | Default Value | Default Hyper-Space |
|---|---|---|---|---|
| 0 | model_class | Model class. Used internally for save/load. Changing this may cause unexpected behaviors. | <class 'matchzoo.models.dense_baseline.DenseBaseline'> | |
| 1 | input_shapes | Dependent on the model and data. Should be set manually. | | |
| 2 | task | Decides model output shape, loss, and metrics. | | |
| 3 | optimizer | | adam | |
| 4 | with_multi_layer_perceptron | A flag of whether a multiple layer perceptron is used. Shouldn't be changed. | True | |
| 5 | mlp_num_units | Number of units in first *mlp_num_layers* layers. | 256 | quantitative uniform distribution in [16, 512), with a step size of 1 |
| 6 | mlp_num_layers | Number of layers of the multiple layer percetron. | 3 | quantitative uniform distribution in [1, 5), with a step size of 1 |
| 7 | mlp_num_fan_out | Number of units of the layer that connects the multiple layer percetron and the output. | 64 | quantitative uniform distribution in [4, 128), with a step size of 4 |
| 8 | mlp_activation_func | Activation function used in the multiple layer perceptron. | relu | |

## 2.5 ArcI

### 2.5.1 Model Documentation

ArcI Model.

**Examples:**

```
>>> model = ArcI()
>>> model.params['num_blocks'] = 1
>>> model.params['left_filters'] = [32]
>>> model.params['right_filters'] = [32]
>>> model.params['left_kernel_sizes'] = [3]
>>> model.params['right_kernel_sizes'] = [3]
>>> model.params['left_pool_sizes'] = [2]
>>> model.params['right_pool_sizes'] = [4]
>>> model.params['conv_activation_func'] = 'relu'
>>> model.params['mlp_num_layers'] = 1
>>> model.params['mlp_num_units'] = 64
```

(continues on next page)

(continued from previous page)

```
>>> model.params['mlp_num_fan_out'] = 32
>>> model.params['mlp_activation_func'] = 'relu'
>>> model.params['dropout_rate'] = 0.5
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
```

### 2.5.2 Model Hyper Parameters

| | Name | Description | Default Value | Default Hyper-Space |
|---|---|---|---|---|
| 0 | model_class | Model class. Used internally for save/load. Changing this may cause unexpected behaviors. | <class 'matchzoo.models.arci.ArcI'> | |
| 1 | input_shapes | Dependent on the model and data. Should be set manually. | | |
| 2 | task | Decides model output shape, loss, and metrics. | | |
| 3 | optimizer | | adam | |
| 4 | with_embedding | A flag used help *auto* module. Shouldn't be changed. | True | |
| 5 | embedding_input_dim | Usually equals vocab size + 1. Should be set manually. | | |
| 6 | embedding_output_dim | Should be set manually. | | |
| 7 | embedding_trainable | *True* to enable embedding layer training, *False* to freeze embedding parameters. | True | |
| 8 | with_multi_layer_perceptron | A flag of whether a multiple layer perceptron is used. Shouldn't be changed. | True | |
| 9 | mlp_num_units | Number of units in first *mlp_num_layers* layers. | 128 | quantitative uniform distribution in [8, 256), with a step size of 8 |
| 10 | mlp_num_layers | Number of layers of the multiple layer percetron. | 3 | quantitative uniform distribution in [1, 6), with a step size of 1 |
| 11 | mlp_num_fan_out | Number of units of the layer that connects the multiple layer percetron and the output. | 64 | quantitative uniform distribution in [4, 128), with a step size of 4 |
| 12 | mlp_activation_func | Activation function used in the multiple layer perceptron. | relu | |
| 13 | num_blocks | Number of convolution blocks. | 1 | |
| 14 | left_filters | The filter size of each convolution blocks for the left input. | [32] | |
| 15 | left_kernel_sizes | The kernel size of each convolution blocks for the left input. | [3] | |
| 16 | right_filters | The filter size of each convolution blocks for the right input. | [32] | |
| 17 | right_kernel_sizes | The kernel size of each convolution blocks for the right input. | [3] | |
| 18 | conv_activation_func | The activation function in the convolution layer. | relu | |
| 19 | left_pool_sizes | The pooling size of each convolution blocks for the left input. | [2] | |
| 20 | right_pool_sizes | The pooling size of each convolution blocks for the right input. | [2] | |
| 21 | padding | The padding mode in the convolution layer. It should be oneof *same*, *valid*, and *causal*. | same | choice in ['same', 'valid', 'causal'] |
| 22 | dropout_rate | The dropout rate. | 0.0 | quantitative uniform distribution in [0.0, 0.8), with a step size of 0.01 |

## 2.6 ArcII

### 2.6.1 Model Documentation

ArcII Model.

**Examples:**

```
>>> model = ArcII()
>>> model.params['embedding_output_dim'] = 300
>>> model.params['num_blocks'] = 2
>>> model.params['kernel_1d_count'] = 32
>>> model.params['kernel_1d_size'] = 3
>>> model.params['kernel_2d_count'] = [16, 32]
>>> model.params['kernel_2d_size'] = [[3, 3], [3, 3]]
>>> model.params['pool_2d_size'] = [[2, 2], [2, 2]]
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
```

### 2.6.2 Model Hyper Parameters

| | Name | Description | Default Value | Default Hyper-Space |
|---|---|---|---|---|
| 0 | model_class | Model class. Used internally for save/load. Changing this may cause unexpected behaviors. | <class 'matchzoo.models.arcii.ArcII'> | |
| 1 | input_shapes | Dependent on the model and data. Should be set manually. | | |
| 2 | task | Decides model output shape, loss, and metrics. | | |
| 3 | optimizer | | adam | choice in ['adam', 'rmsprop', 'adagrad'] |
| 4 | with_embedding | A flag used help *auto* module. Shouldn't be changed. | True | |
| 5 | embedding_input_dim | Usually equals vocab size + 1. Should be set manually. | | |
| 6 | embedding_output_dim | Should be set manually. | | |
| 7 | embedding_trainable | *True* to enable embedding layer training, *False* to freeze embedding parameters. | True | |
| 8 | num_blocks | Number of 2D convolution blocks. | 1 | |
| 9 | kernel_1d_count | Kernel count of 1D convolution layer. | 32 | |
| 10 | kernel_1d_size | Kernel size of 1D convolution layer. | 3 | |
| 11 | kernel_2d_count | Kernel count of 2D convolution layer in each block | [32] | |
| 12 | kernel_2d_size | Kernel size of 2D convolution layer in each block. | [[3, 3]] | |
| 13 | activation | Activation function. | relu | |
| 14 | pool_2d_size | Size of pooling layer in each block. | [[2, 2]] | |
| 15 | padding | The padding mode in the convolution layer. It should be oneof *same*, *valid*. | same | choice in ['same', 'valid'] |
| 16 | dropout_rate | The dropout rate. | 0.0 | quantitative uniform distribution in [0.0, 0.8), with a step size of 0.01 |

## 2.7 MatchPyramid

### 2.7.1 Model Documentation

MatchPyramid Model.

**Examples:**

```
>>> model = MatchPyramid()
>>> model.params['embedding_output_dim'] = 300
>>> model.params['num_blocks'] = 2
>>> model.params['kernel_count'] = [16, 32]
>>> model.params['kernel_size'] = [[3, 3], [3, 3]]
>>> model.params['dpool_size'] = [3, 10]
```

(continues on next page)

(continued from previous page)

```
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
```

### 2.7.2 Model Hyper Parameters

| | Name | Description | Default Value | Default Hyper-Space |
|---|---|---|---|---|
| 0 | model_class | Model class. Used internally for save/load. Changing this may cause unexpected behaviors. | <class 'matchzoo.models.match_pyramid.MatchPyramid'> | |
| 1 | input_shapes | Dependent on the model and data. Should be set manually. | | |
| 2 | task | Decides model output shape, loss, and metrics. | | |
| 3 | optimizer | | adam | |
| 4 | with_embedding | A flag used help *auto* module. Shouldn't be changed. | True | |
| 5 | embedding_input_dim | Usually equals vocab size + 1. Should be set manually. | | |
| 6 | embedding_output_dim | Should be set manually. | | |
| 7 | embedding_trainable | *True* to enable embedding layer training, *False* to freeze embedding parameters. | True | |
| 8 | num_blocks | Number of convolution blocks. | 1 | |
| 9 | kernel_count | The kernel count of the 2D convolution of each block. | [32] | |
| 10 | kernel_size | The kernel size of the 2D convolution of each block. | [[3, 3]] | |
| 11 | activation | The activation function. | relu | |
| 12 | dpool_size | The max-pooling size of each block. | [3, 10] | |
| 13 | padding | The padding mode in the convolution layer. | same | |
| 14 | dropout_rate | The dropout rate. | 0.0 | quantitative uniform distribution in [0.0, 0.8), with a step size of 0.01 |

## 2.8 KNRM

### 2.8.1 Model Documentation

KNRM model.

**Examples:**

```
>>> model = KNRM()
>>> model.params['embedding_input_dim'] =  10000
>>> model.params['embedding_output_dim'] =  10
```

(continues on next page)

(continued from previous page)

```
>>> model.params['embedding_trainable'] = True
>>> model.params['kernel_num'] = 11
>>> model.params['sigma'] = 0.1
>>> model.params['exact_sigma'] = 0.001
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
```

### 2.8.2 Model Hyper Parameters

| | Name | Description | Default Value | Default Hyper-Space |
|---|---|---|---|---|
| 0 | model_class | Model class. Used internally for save/load. Changing this may cause unexpected behaviors. | <class 'matchzoo.models.knrm.KNRM'> | |
| 1 | input_shapes | Dependent on the model and data. Should be set manually. | | |
| 2 | task | Decides model output shape, loss, and metrics. | | |
| 3 | optimizer | | adam | |
| 4 | with_embedding | A flag used help *auto* module. Shouldn't be changed. | True | |
| 5 | embedding_input_dim | Usually equals vocab size + 1. Should be set manually. | | |
| 6 | embedding_output_dim | Should be set manually. | | |
| 7 | embedding_trainable | *True* to enable embedding layer training, *False* to freeze embedding parameters. | True | |
| 8 | kernel_num | The number of RBF kernels. | 11 | quantitative uniform distribution in [5, 20), with a step size of 1 |
| 9 | sigma | The *sigma* defines the kernel width. | 0.1 | quantitative uniform distribution in [0.01, 0.2), with a step size of 0.01 |
| 10 | exact_sigma | The *exact_sigma* denotes the *sigma* for exact match. | 0.001 | |

## 2.9 DUET

### 2.9.1 Model Documentation

DUET Model.

**Examples:**

```
>>> model = DUET()
>>> model.params['embedding_input_dim'] = 1000
>>> model.params['embedding_output_dim'] = 300
>>> model.params['lm_filters'] = 32
>>> model.params['lm_hidden_sizes'] = [64, 32]
>>> model.params['dropout_rate'] = 0.5
```

(continues on next page)

(continued from previous page)

```
>>> model.params['dm_filters'] = 32
>>> model.params['dm_kernel_size'] = 3
>>> model.params['dm_d_mpool'] = 4
>>> model.params['dm_hidden_sizes'] = [64, 32]
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
```

### 2.9.2 Model Hyper Parameters

| | Name | Description | Default Value | Default Hyper-Space |
|---|---|---|---|---|
| 0 | model_class | Model class. Used internally for save/load. Changing this may cause unexpected behaviors. | <class 'matchzoo.models.duet.DUET'> | |
| 1 | input_shapes | Dependent on the model and data. Should be set manually. | | |
| 2 | task | Decides model output shape, loss, and metrics. | | |
| 3 | optimizer | | adam | |
| 4 | with_embedding | A flag used help *auto* module. Shouldn't be changed. | True | |
| 5 | embedding_input_dim | Usually equals vocab size + 1. Should be set manually. | | |
| 6 | embedding_output_dim | Should be set manually. | | |
| 7 | embedding_trainable | *True* to enable embedding layer training, *False* to freeze embedding parameters. | True | |
| 8 | lm_filters | Filter size of 1D convolution layer in the local model. | 32 | |
| 9 | lm_hidden_sizes | A list of hidden size of the MLP layer in the local model. | [32] | |
| 10 | dm_filters | Filter size of 1D convolution layer in the distributed model. | 32 | |
| 11 | dm_kernel_size | Kernel size of 1D convolution layer in the distributed model. | 3 | |
| 12 | dm_q_hidden_size | Hidden size of the MLP layer for the left text in the distributed model. | 32 | |
| 13 | dm_d_mpool | Max pooling size for the right text in the distributed model. | 3 | |
| 14 | dm_hidden_sizes | A list of hidden size of the MLP layer in the distributed model. | [32] | |
| 15 | padding | The padding mode in the convolution layer. It should be one of *same*, *valid*, and *causal*. | same | |
| 16 | activation_func | Activation function in the convolution layer. | relu | |
| 17 | dropout_rate | The dropout rate. | 0.5 | quantitative uniform distribution in [0.0, 0.8), with a step size of 0.02 |

## 2.10 DRMMTKS

### 2.10.1 Model Documentation

DRMMTKS Model.

**Examples:**

```
>>> model = DRMMTKS()
>>> model.params['embedding_input_dim'] = 10000
>>> model.params['embedding_output_dim'] = 100
>>> model.params['top_k'] = 20
>>> model.params['mlp_num_layers'] = 1
>>> model.params['mlp_num_units'] = 5
>>> model.params['mlp_num_fan_out'] = 1
>>> model.params['mlp_activation_func'] = 'tanh'
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
```

### 2.10.2 Model Hyper Parameters

| | Name | Description | Default Value | Default Hyper-Space |
|---|---|---|---|---|
| 0 | model_class | Model class. Used internally for save/load. Changing this may cause unexpected behaviors. | <class 'matchzoo.models.drmmtks.DRMMTKS'> | |
| 1 | input_shapes | Dependent on the model and data. Should be set manually. | [(5,), (300,)] | |
| 2 | task | Decides model output shape, loss, and metrics. | | |
| 3 | optimizer | | adam | |
| 4 | with_embedding | A flag used help *auto* module. Shouldn't be changed. | True | |
| 5 | embedding_input_dim | Usually equals vocab size + 1. Should be set manually. | | |
| 6 | embedding_output_dim | Should be set manually. | | |
| 7 | embedding_trainable | *True* to enable embedding layer training, *False* to freeze embedding parameters. | True | |
| 8 | with_multi_layer_perceptron | A flag of whether a multiple layer perceptron is used. Shouldn't be changed. | True | |
| 9 | mlp_num_units | Number of units in first *mlp_num_layers* layers. | 128 | quantitative uniform distribution in [8, 256), with a step size of 8 |
| 10 | mlp_num_layers | Number of layers of the multiple layer percetron. | 3 | quantitative uniform distribution in [1, 6), with a step size of 1 |
| 11 | mlp_num_fan_out | Number of units of the layer that connects the multiple layer percetron and the output. | 64 | quantitative uniform distribution in [4, 128), with a step size of 4 |
| 12 | mlp_activation_func | Activation function used in the multiple layer perceptron. | relu | |
| 13 | mask_value | The value to be masked from inputs. | -1 | |
| 14 | top_k | Size of top-k pooling layer. | 10 | quantitative uniform distribution in [2, 100), with a step size of 1 |

## 2.11 DRMM

### 2.11.1 Model Documentation

DRMM Model.

**Examples:**

```
>>> model = DRMM()
>>> model.params['mlp_num_layers'] = 1
>>> model.params['mlp_num_units'] = 5
>>> model.params['mlp_num_fan_out'] = 1
>>> model.params['mlp_activation_func'] = 'tanh'
```

(continues on next page)

(continued from previous page)

```
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
>>> model.compile()
```

### 2.11.2 Model Hyper Parameters

| | Name | Description | Default Value | Default Hyper-Space |
|---|---|---|---|---|
| 0 | model_class | Model class. Used internally for save/load. Changing this may cause unexpected behaviors. | <class 'matchzoo.models.drmm.DRMM'> | |
| 1 | input_shapes | Dependent on the model and data. Should be set manually. | [(5,), (5, 30)] | |
| 2 | task | Decides model output shape, loss, and metrics. | | |
| 3 | optimizer | | adam | |
| 4 | with_embedding | A flag used help *auto* module. Shouldn't be changed. | True | |
| 5 | embedding_input_dim | Usually equals vocab size + 1. Should be set manually. | | |
| 6 | embedding_output_dim | Should be set manually. | | |
| 7 | embedding_trainable | *True* to enable embedding layer training, *False* to freeze embedding parameters. | True | |
| 8 | with_multi_layer_perceptron | A flag of whether a multiple layer perceptron is used. Shouldn't be changed. | True | |
| 9 | mlp_num_units | Number of units in first *mlp_num_layers* layers. | 128 | quantitative uniform distribution in [8, 256), with a step size of 8 |
| 10 | mlp_num_layers | Number of layers of the multiple layer percetron. | 3 | quantitative uniform distribution in [1, 6), with a step size of 1 |
| 11 | mlp_num_fan_out | Number of units of the layer that connects the multiple layer percetron and the output. | 64 | quantitative uniform distribution in [4, 128), with a step size of 4 |
| 12 | mlp_activation_func | Activation function used in the multiple layer perceptron. | relu | |
| 13 | mask_value | The value to be masked from inputs. | -1 | |

## 2.12 ANMM

### 2.12.1 Model Documentation

ANMM Model.

**Examples:**

```
>>> model = ANMM()
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
```

### 2.12.2 Model Hyper Parameters

| | Name | Description | Default Value | Default Hyper-Space |
|---|---|---|---|---|
| 0 | model_class | Model class. Used internally for save/load. Changing this may cause unexpected behaviors. | <class 'matchzoo.models.anmm.ANMM'> | |
| 1 | input_shapes | Dependent on the model and data. Should be set manually. | | |
| 2 | task | Decides model output shape, loss, and metrics. | | |
| 3 | optimizer | | adam | |
| 4 | with_embedding | A flag used help *auto* module. Shouldn't be changed. | True | |
| 5 | embedding_input_dim | Usually equals vocab size + 1. Should be set manually. | | |
| 6 | embedding_output_dim | Should be set manually. | | |
| 7 | embedding_trainable | *True* to enable embedding layer training, *False* to freeze embedding parameters. | True | |
| 8 | dropout_rate | The dropout rate. | 0.1 | quantitative uniform distribution in [0, 1), with a step size of 0.05 |
| 9 | num_layers | Number of hidden layers in the MLP layer. | 2 | |
| 10 | hidden_sizes | Number of hidden size for each hidden layer | [30, 30] | |

## 2.13 MVLSTM

### 2.13.1 Model Documentation

MVLSTM Model.

**Examples:**

```
>>> model = MVLSTM()
>>> model.params['lstm_units'] = 32
>>> model.params['top_k'] = 50
>>> model.params['mlp_num_layers'] = 2
>>> model.params['mlp_num_units'] = 20
>>> model.params['mlp_num_fan_out'] = 10
>>> model.params['mlp_activation_func'] = 'relu'
>>> model.params['dropout_rate'] = 0.5
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
```

### 2.13.2 Model Hyper Parameters

| | Name | Description | Default Value | Default Hyper-Space |
|---|---|---|---|---|
| 0 | model_class | Model class. Used internally for save/load. Changing this may cause unexpected behaviors. | <class 'matchzoo.models.mvlstm.MVLSTM'> | |
| 1 | input_shapes | Dependent on the model and data. Should be set manually. | | |
| 2 | task | Decides model output shape, loss, and metrics. | | |
| 3 | optimizer | | adam | |
| 4 | with_embedding | A flag used help *auto* module. Shouldn't be changed. | True | |
| 5 | embedding_input_dim | Usually equals vocab size + 1. Should be set manually. | | |
| 6 | embedding_output_dim | Should be set manually. | | |
| 7 | embedding_trainable | *True* to enable embedding layer training, *False* to freeze embedding parameters. | True | |
| 8 | with_multi_layer_perceptron | A flag of whether a multiple layer perceptron is used. Shouldn't be changed. | True | |
| 9 | mlp_num_units | Number of units in first *mlp_num_layers* layers. | 128 | quantitative uniform distribution in [8, 256), with a step size of 8 |
| 10 | mlp_num_layers | Number of layers of the multiple layer percetron. | 3 | quantitative uniform distribution in [1, 6), with a step size of 1 |
| 11 | mlp_num_fan_out | Number of units of the layer that connects the multiple layer percetron and the output. | 64 | quantitative uniform distribution in [4, 128), with a step size of 4 |
| 12 | mlp_activation_func | Activation function used in the multiple layer perceptron. | relu | |
| 13 | lstm_units | Integer, the hidden size in the bi-directional LSTM layer. | 32 | |
| 14 | dropout_rate | Float, the dropout rate. | 0.0 | |
| 15 | top_k | Integer, the size of top-k pooling layer. | 10 | quantitative uniform distribution in [2, 100), with a step size of 1 |

## 2.14 MatchLSTM

### 2.14.1 Model Documentation

Match LSTM model.

**Examples:**

```
>>> model = MatchLSTM()
>>> model.guess_and_fill_missing_params(verbose=0)
```

(continues on next page)

(continued from previous page)

```
>>> model.params['embedding_input_dim'] = 10000
>>> model.params['embedding_output_dim'] = 100
>>> model.params['embedding_trainable'] = True
>>> model.params['fc_num_units'] = 200
>>> model.params['lstm_num_units'] = 256
>>> model.params['dropout_rate'] = 0.5
>>> model.build()
```

### 2.14.2 Model Hyper Parameters

| | Name | Description | Default Value | Default Hyper-Space |
|---|---|---|---|---|
| 0 | model_class | Model class. Used internally for save/load. Changing this may cause unexpected behaviors. | <class 'matchzoo.contrib.models.match_lstm.MatchLSTM'> | |
| 1 | input_shapes | Dependent on the model and data. Should be set manually. | | |
| 2 | task | Decides model output shape, loss, and metrics. | | |
| 3 | optimizer | | adam | |
| 4 | with_embedding | A flag used help *auto* module. Shouldn't be changed. | True | |
| 5 | embedding_input_dim | Usually equals vocab size + 1. Should be set manually. | | |
| 6 | embedding_output_dim | Should be set manually. | | |
| 7 | embedding_trainable | *True* to enable embedding layer training, *False* to freeze embedding parameters. | True | |
| 8 | lstm_num_units | The hidden size in the LSTM layer. | 256 | quantitative uniform distribution in [128, 384), with a step size of 32 |
| 9 | fc_num_units | The hidden size in the full connection layer. | 200 | quantitative uniform distribution in [100, 300), with a step size of 20 |
| 10 | dropout_rate | The dropout rate. | 0.0 | quantitative uniform distribution in [0.0, 0.9), with a step size of 0.01 |

## 2.15 ConvKNRM

### 2.15.1 Model Documentation

ConvKNRM model.

**Examples:**

```
>>> model = ConvKNRM()
>>> model.params['embedding_input_dim'] = 10000
>>> model.params['embedding_output_dim'] = 300
```

(continues on next page)

(continued from previous page)

```
>>> model.params['embedding_trainable'] = True
>>> model.params['filters'] = 128
>>> model.params['conv_activation_func'] = 'tanh'
>>> model.params['max_ngram'] = 3
>>> model.params['use_crossmatch'] = True
>>> model.params['kernel_num'] = 11
>>> model.params['sigma'] = 0.1
>>> model.params['exact_sigma'] = 0.001
>>> model.guess_and_fill_missing_params(verbose=0)
>>> model.build()
```

### 2.15.2 Model Hyper Parameters

| | Name | Description | Default Value | Default Hyper-Space |
|---|---|---|---|---|
| 0 | model_class | Model class. Used internally for save/load. Changing this may cause unexpected behaviors. | <class 'matchzoo.models.conv_knrm.ConvKNRM'> | |
| 1 | input_shapes | Dependent on the model and data. Should be set manually. | | |
| 2 | task | Decides model output shape, loss, and metrics. | | |
| 3 | optimizer | | adam | |
| 4 | with_embedding | A flag used help *auto* module. Shouldn't be changed. | True | |
| 5 | embedding_input_dim | Usually equals vocab size + 1. Should be set manually. | | |
| 6 | embedding_output_dim | Should be set manually. | | |
| 7 | embedding_trainable | *True* to enable embedding layer training, *False* to freeze embedding parameters. | True | |
| 8 | kernel_num | The number of RBF kernels. | 11 | quantitative uniform distribution in [5, 20), with a step size of 1 |
| 9 | sigma | The *sigma* defines the kernel width. | 0.1 | quantitative uniform distribution in [0.01, 0.2), with a step size of 0.01 |
| 10 | exact_sigma | The *exact_sigma* denotes the *sigma* for exact match. | 0.001 | |
| 11 | filters | The filter size in the convolution layer. | 128 | |
| 12 | conv_activation_func | The activation function in the convolution layer. | relu | |
| 13 | max_ngram | The maximum length of n-grams for the convolution layer. | 3 | |
| 14 | use_crossmatch | Whether to match left n-grams and right n-grams of different lengths | True | |

CHAPTER 3

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## m

# Index

## C



## D



## E

## F



## G



## H

## N



## O

## P



## Q



## R



## S

## T



## U



## V

## W