Snorkel Documentation

Release 0.6.0

Alex Ratner, Stephen Bach, Henry Ehrenberg

Apr 06, 2018

Contents

snorkel

CHAPTER 1

Contexts

Preprocessed input data is represented in Snorkel as a hierarchy of *Context* subclass objects. For example, as is currently default for text: Corpus -> Document -> Sentence -> Span.

1.1 Core Data Models

class snorkel.models.context.**Context**(***kwargs*)
A piece of content from which Candidates are composed.

class snorkel.models.context.**Document**(***kwargs*)
A root Context.

class snorkel.models.context.**Sentence**(***kwargs*)
A sentence Context in a Document.

class snorkel.models.context.**Span**(***kwargs*)
A span of characters, identified by Context id and character-index start, end (inclusive).

char_offsets are **relative to the Context start**

class snorkel.models.context.**TemporaryContext**
A context which does not incur the overhead of a proper ORM-based Context object. The TemporaryContext class is specifically for the candidate extraction process, during which a CandidateSpace object will generate many TemporaryContexts, which will then be filtered by Matchers prior to materialization of Candidates and constituent Context objects.

Every Context object has a corresponding TemporaryContext object from which it inherits.

A TemporaryContext must have specified equality / set membership semantics, a stable_id for checking uniqueness against the database, and a promote() method which returns a corresponding Context object.

class snorkel.models.context.**TemporarySpan**(*sentence*, *char_start*, *char_end*, *meta=None*)
The TemporaryContext version of Span

char_to_word_index(*ci*)
Given a character-level index (offset), return the index of the **word this char is in**

get_attrib_span(*a*, *sep=u' '*)
Get the span of sentence attribute _a_ over the range defined by word_offset, n

get_attrib_tokens(*a=u'words'*)
Get the tokens of sentence attribute _a_ over the range defined by word_offset, n

word_to_char_index(*wi*)
Given a word-level index, return the character-level index (offset) of the word's start

snorkel.models.context.**construct_stable_id**(*parent_context*, *polymorphic_type*, *relative_char_offset_start*, *relative_char_offset_end*)
Contruct a stable ID for a Context given its parent and its character offsets relative to the parent

snorkel.models.context.**split_stable_id**(*stable_id*)

Split stable id, returning:

- Document (root) stable ID
- Context polymorphic type
- Character offset start, end *relative to document start*

Returns tuple of four values.

1.2 Core Objects for Preprocessing and Loading

class snorkel.parser.doc_preprocessors.**CSVPathsPreprocessor**(*path*, *parser_factory=<class 'snorkel.parser.doc_preprocessors.TextDocPr*, *column=None*, *delim=u','*, **args*, ***kwargs*)
This *DocumentPreprocessor* treats inputs file as index of paths to actual documents; each line in the input file contains a path to a document.

Defaults and Customization:

- The input file is treated as a simple text file having one path per file. However, if the input is a CSV file, a pair of `column` and `delim` parameters may be used to retrieve the desired value as reference path.
- The referenced documents are treated as text document and hence parsed using `TextDocPreprocessor`. However, if the referenced files are complex, an advanced parser may be used by specifying `parser_factory` parameter to constructor.

class snorkel.parser.doc_preprocessors.**DocPreprocessor**(*path*, *encoding=u'utf-8'*, *max_docs=inf*)
Processes a file or directory of files into a set of Document objects.

Parameters

- **encoding** – file encoding to use, default='utf-8'
- **path** – filesystem path to file or directory to parse
- **max_docs** – the maximum number of Documents to produce, default=float('inf')

generate()
Parses a file or directory of files into a set of Document objects.

class snorkel.parser.doc_preprocessors.**HTMLDocPreprocessor**(*path*, *encoding=u'utf-8'*, *max_docs=inf*)

Simple parsing of raw HTML files, assuming one document per file

class snorkel.parser.doc_preprocessors.**TSVDocPreprocessor**(*path*, *encoding=u'utf-8'*, *max_docs=inf*)

Simple parsing of TSV file with one (doc_name <tab> doc_text) per line

class snorkel.parser.doc_preprocessors.**TextDocPreprocessor**(*path*, *encoding=u'utf-8'*, *max_docs=inf*)

Simple parsing of raw text files, assuming one document per file

class snorkel.parser.doc_preprocessors.**TikaPreprocessor**(*path*, *encoding=u'utf-8'*, *max_docs=inf*)

This preprocessor use Apache Tika parser to retrieve text content from complex file types such as DOCX, HTML and PDFs.

Documentation for customizing Tika is here

Example:

```
!find pdf_dir -name *.pdf > input.csv # list of files
from snorkel.parser import (
    TikaPreprocessor, CSVPathsPreprocessor, CorpusParser
)
CorpusParser().apply(
    CSVPathsPreprocessor('input.csv', parser_factory=TikaPreprocessor)
)
```

class snorkel.parser.doc_preprocessors.**XMLMultiDocPreprocessor**(*path*, *doc=u'.//document'*, *text=u'./text/text()'*, *id=u'./id/text()'*, *keep_xml_tree=False*, **args*, ***kwargs*)

Parse an XML file _which contains multiple **documents_** into a set of Document objects.

Use XPath queries to specify a _document_ object, and then for each document, a set of _text_ sections and an _id_.

Note: Include the full document XML etree in the attribs dict with keep_xml_tree=True

class snorkel.parser.parser.**ParserConnection**(*parser*)

Default connection object assumes local parser object

class snorkel.parser.parser.**URLParserConnection**(*parser*, *retries=5*)

URL parser connection

parse(*document*, *text*)

Return parse generator :param document: :param text: :return:

post(*url*, *data*, *allow_redirects=True*)

Parameters

- **url** –
- **data** –
- **allow_redirects** –
- **timeout** –

Returns

CHAPTER 2

Candidates

In order to apply machine learning–i.e., in this case, a classifier–to information extraction problems, we need to have a base set of objects that are being classified. In Snorkel, these are the *Candidate* subclasses, which are defined over *Context* arguments, and represent *potential* mentions to extract. We use *Matcher* operators to extract a set of *Candidate* objects from the input data.

2.1 Core Data Models

class snorkel.models.candidate.**Candidate**(***kwargs*)
An abstract candidate relation.

New relation types should be defined by calling candidate_subclass(), **not** subclassing this class directly.

get_cids()
Get a tuple of the canonical IDs (CIDs) of the contexts making up this candidate

get_contexts()
Get a tuple of the consituent contexts making up this candidate

class snorkel.models.candidate.**Marginal**(***kwargs*)
A marginal probability corresponding to a (Candidate, value) pair.

Represents:

P(candidate = value) = probability

@training: If True, this is a training marginal; otherwise is end prediction

snorkel.models.candidate.**candidate_subclass**(*class_name*, *args*, *table_name=None*, *cardinality=None*, *values=None*)
Creates and returns a Candidate subclass with provided argument names, which are Context type. Creates the table in DB if does not exist yet.

Import using:

```
from snorkel.models import candidate_subclass
```

Parameters

- **class_name** – The name of the class, should be "camel case" e.g. NewCandidate
- **args** – A list of names of consituent arguments, which refer to the Contexts–representing mentions–that comprise the candidate
- **table_name** – The name of the corresponding table in DB; if not provided, is converted from camel case by default, e.g. new_candidate
- **cardinality** – The cardinality of the variable corresponding to the Candidate. By default is 2 i.e. is a binary value, e.g. is or is not a true mention.

2.2 Core Objects for Candidate Extraction

class snorkel.candidates.**CandidateExtractor**(*candidate_class*, *cspaces*, *matchers*, *self_relations=False*, *nested_relations=False*, *symmetric_relations=False*)

An operator to extract Candidate objects from a Context.

Parameters

- **candidate_class** – The type of relation to extract, defined using *snorkel.models.candidate_subclass*
- **cspaces** – one or list of *CandidateSpace* objects, one for each relation argument. Defines space of Contexts to consider
- **matchers** – one or list of *snorkel.matchers.Matcher* objects, one for each relation argument. Only tuples of Contexts for which each element is accepted by the corresponding Matcher will be returned as Candidates
- **self_relations** – Boolean indicating whether to extract Candidates that relate the same context. Only applies to binary relations. Default is False.
- **nested_relations** – Boolean indicating whether to extract Candidates that relate one Context with another that contains it. Only applies to binary relations. Default is False.
- **symmetric_relations** – Boolean indicating whether to extract symmetric Candidates, i.e., rel(A,B) and rel(B,A), where A and B are Contexts. Only applies to binary relations. Default is False.

class snorkel.candidates.**CandidateSpace**

Defines the **space** of candidate objects Calling _apply(x)_ given an object _x_ returns a generator over candidates in _x_.

class snorkel.candidates.**Ngrams**(*n_max=5*, *split_tokens=(u'-', u'/')*)

Defines the space of candidates as all n-grams (n <= n_max) in a Sentence _x_, indexing by **character offset**.

class snorkel.candidates.**PretaggedCandidateExtractor**(*candidate_class*, *entity_types*, *self_relations=False*, *nested_relations=False*, *symmetric_relations=True*, *entity_sep=u'~@~'*)

UDFRunner for PretaggedCandidateExtractorUDF

class snorkel.candidates.**PretaggedCandidateExtractorUDF**(*candidate_class*, *entity_types*, *self_relations=False*, *nested_relations=False*, *symmetric_relations=False*, *entity_sep=u'~@~'*, ***kwargs*)

An extractor for Sentences with entities pre-tagged, and stored in the entity_types and entity_cids fields.

apply(*context*, *clear*, *split*, *check_for_existing=True*, ***kwargs*)

Extract Candidates from a Context

class snorkel.matchers.**Concat**(**children*, ***opts*)

Selects candidates which are the concatenation of adjacent matches from child operators NOTE: Currently slices on **word index** and considers concatenation along these divisions only

class snorkel.matchers.**DateMatcher**(**children*, ***kwargs*)

Matches Spans that are dates, as identified by CoreNLP.

A convenience class for setting up a RegexMatchEach to match spans for which each token was tagged as a date.

class snorkel.matchers.**DictionaryMatch**(**children*, ***opts*)

Selects candidate Ngrams that match against a given list d

class snorkel.matchers.**LambdaFunctionMatcher**(**children*, ***opts*)

Selects candidate Ngrams that return True when fed to a function f.

class snorkel.matchers.**LocationMatcher**(**children*, ***kwargs*)

Matches Spans that are the names of locations, as identified by CoreNLP.

A convenience class for setting up a RegexMatchEach to match spans for which each token was tagged as a location.

class snorkel.matchers.**Matcher**(**children*, ***opts*)

Applies a function f : c -> {True,False} to a generator of candidates, returning only candidates _c_ s.t. _f(c) == True_, where f can be compositionally defined.

apply(*candidates*)

Apply the Matcher to a **generator** of candidates Optionally only takes the longest match (NOTE: assumes this is the *first* match)

f(*c*)

The recursively composed version of filter function f By default, returns logical **conjunction** of operator and single child operator

class snorkel.matchers.**MiscMatcher**(**children*, ***kwargs*)

Matches Spans that are miscellaneous named entities, as identified by CoreNLP.

A convenience class for setting up a RegexMatchEach to match spans for which each token was tagged as miscellaneous.

class snorkel.matchers.**NgramMatcher**(**children*, ***opts*)

Matcher base class for Ngram objects

class snorkel.matchers.**NumberMatcher**(**children*, ***kwargs*)

Matches Spans that are numbers, as identified by CoreNLP.

A convenience class for setting up a RegexMatchEach to match spans for which each token was tagged as a number.

class snorkel.matchers.**OrganizationMatcher**(**children*, ***kwargs*)
Matches Spans that are the names of organizations, as identified by CoreNLP.

A convenience class for setting up a RegexMatchEach to match spans for which each token was tagged as an organization.

class snorkel.matchers.**PersonMatcher**(**children*, ***kwargs*)
Matches Spans that are the names of people, as identified by CoreNLP.

A convenience class for setting up a RegexMatchEach to match spans for which each token was tagged as a person.

class snorkel.matchers.**RegexMatch**(**children*, ***opts*)
Base regex class- does not specify specific semantics of *what* is being matched yet

class snorkel.matchers.**RegexMatchEach**(**children*, ***opts*)
Matches regex pattern on **each token**

class snorkel.matchers.**RegexMatchSpan**(**children*, ***opts*)
Matches regex pattern on **full concatenated span**

class snorkel.matchers.**SlotFillMatch**(**children*, ***opts*)
Matches a slot fill pattern of matchers _at the character **level_**

class snorkel.matchers.**Union**(**children*, ***opts*)
Takes the union of candidate sets returned by child operators

CHAPTER 3

Annotations

One of the core operations in Snorkel is _annotating_ the candidates in various ways. We can think of generating features for the candidates as annotating them (creating a *Feature* object), and can also view supervision via labeling functions as annotating them (creating a *Label* object).

3.1 Core Data Models

class snorkel.models.annotation.**AnnotationKeyMixin**
Mixin class for defining annotation key tables. An AnnotationKey is the unique name associated with a set of Annotations, corresponding e.g. to a single labeling or feature function. An AnnotationKey may have an associated weight (Parameter) associated with it.

class snorkel.models.annotation.**AnnotationMixin**
Mixin class for defining annotation tables. An annotation is a value associated with a Candidate. Examples include labels, features, and predictions. New types of annotations can be defined by creating an annotation class and corresponding annotation, for example:

```
from snorkel.models.annotation import AnnotationMixin
from snorkel.models.meta import SnorkelBase

class NewAnnotation(AnnotationMixin, SnorkelBase):
    value = Column(Float, nullable=False)


# The entire storage schema, including NewAnnotation, can now be initialized with␣
↪the following import
import snorkel.models
```

The annotation class should include a Column attribute named value.

class snorkel.models.annotation.**Feature**(***kwargs*)
An element of a representation of a Candidate in a feature space.

A Feature's annotation key identifies the definition of the Feature, e.g., a function that implements it or the library name and feature name in an automatic featurization library.

class snorkel.models.annotation.**GoldLabel**(***kwargs*)
A separate class for labels from human annotators or other gold standards.

class snorkel.models.annotation.**Label**(***kwargs*)
A discrete label associated with a Candidate, indicating a target prediction value.

Labels are used to represent the output of labeling functions.

A Label's annotation key identifies the labeling function that provided the Label.

class snorkel.models.annotation.**Prediction**(***kwargs*)
A probability associated with a Candidate, indicating the degree of belief that the Candidate is true.

A Prediction's annotation key indicates which process or method produced the Prediction, e.g., which model with which ParameterSet.

class snorkel.models.annotation.**StableLabel**(***kwargs*)
A special secondary table for preserving labels created by *human annotators* (e.g. in the Viewer) in a stable format that does not cascade, and is independent of the Candidate ids.

3.2 Core Objects for Annotations (Features, Labels)

class snorkel.annotations.**Annotator**(*annotation_class*, *annotation_key_class*, *f_gen*)
Abstract class for annotating candidates and persisting these annotations to DB

apply_existing(*split=0*, *key_group=0*, *cids_query=None*, ***kwargs*)
Alias for apply that emphasizes we are using an existing AnnotatorKey set.

clear(*session*, *split=0*, *key_group=0*, *replace_key_set=True*, *cids_query=None*, ***kwargs*)
Deletes the Annotations for the Candidates in the given split. If replace_key_set=True, deletes *all* Annotations (of this Annotation sub-class) and also deletes all AnnotationKeys (of this sub-class)

class snorkel.annotations.**FeatureAnnotator**(*f=<function get_span_feats>*)
Apply feature generators to the candidates, generating Feature annotations

class snorkel.annotations.**LabelAnnotator**(*lfs=None*, *label_generator=None*)
Apply labeling functions to the candidates, generating Label annotations

Parameters **lfs** – A _list_ of labeling functions (LFs)

snorkel.annotations.**load_marginals**(*session*, *X=None*, *split=0*, *cids_query=None*, *training=True*)
Load the marginal probs. for a given split of Candidates

snorkel.annotations.**load_matrix**(*matrix_class*, *annotation_key_class*, *annotation_class*, *session*, *split=0*, *cids_query=None*, *key_group=0*, *key_names=None*, *zero_one=False*, *load_as_array=False*, *coerce_int=True*)
Returns the annotations corresponding to a split of candidates with N members and an AnnotationKey group with M distinct keys as an N x M CSR sparse matrix.

snorkel.annotations.**save_marginals**(*session*, *X*, *marginals*, *training=True*)
Save marginal probabilities for a set of Candidates to db.

Parameters

- **X** – Either an M x N csr_AnnotationMatrix-class matrix, where M is number of candidates, N number of LFs/features; OR a list of arbitrary objects with candidate ids accessible via a .id attrib

- **marginals** – A dense M x K matrix of marginal probabilities, where K is the cardinality of the candidates, OR a M-dim list/array if K=2.
- **training** – If True, these are training marginals / labels; else they are saved as end model predictions.

Note: The marginals for k=0 are not stored, only for k = 1,. . . ,K

CHAPTER 4

Learning

In the Snorkel pipeline, the user writes labeling functions (LFs) and then uses the generative model to unify and denoise their labels. Then, the marginal predictions of this model are used as probabilistic training labels for the discriminative model. Currently we provide bindings for TensorFlow models, and two basic models: Logistic regression and an LSTM. See tutorials for a more in-depth explanation.

4.1 Base Classifier Class

class snorkel.learning.classifier.**Classifier**(*cardinality=2*, *name=None*)

Simple abstract base class for a probabilistic classifier.

error_analysis(*session*, *X_test*, *Y_test*, *gold_candidate_set=None*, *b=0.5*, *set_unlabeled_as_neg=True*, *display=True*, *scorer=<class 'snorkel.learning.utils.MentionScorer'>*, ***kwargs*)

Prints full score analysis using the Scorer class, and then returns the a tuple of sets conatining the test candidates bucketed for error analysis, i.e.:

- For binary: TP, FP, TN, FN
- For categorical: correct, incorrect

Parameters

- **X_test** – The input test candidates, as a list or annotation matrix
- **Y_test** – The input test labels, as a list or annotation matrix
- **gold_candidate_set** – Full set of TPs in the test set
- **b** – Decision boundary *for binary setting only*
- **set_unlabeled_as_neg** – Whether to map 0 labels -> -1, *binary setting*
- **display** – Print score report
- **scorer** – The Scorer sub-class to use

load()

marginals(*X*, *batch_size=None*, ***kwargs*)

next()

predictions(*X*, *b=0.5*, *batch_size=None*)
Return numpy array of elements in {-1,0,1} based on predicted marginal probabilities.

representation = False

save()

save_marginals(*session*, *X*, *training=False*)
Save the predicted marginal probabilities for the Candidates X.

score(*X_test*, *Y_test*, *b=0.5*, *set_unlabeled_as_neg=True*, *beta=1*, *batch_size=None*)

Returns the summary scores:

- For binary: precision, recall, F-beta score
- For categorical: accuracy

Parameters

- **X_test** – The input test candidates, as a list or annotation matrix
- **Y_test** – The input test labels, as a list or annotation matrix
- **b** – Decision boundary *for binary setting only*
- **set_unlabeled_as_neg** – Whether to map 0 labels -> -1, *binary setting.*
- **beta** – For F-beta score; by default beta = 1 => F-1 score.

Note: Unlike in self.error_analysis, this method assumes X_test and Y_test are properly collated!

4.2 Generative Model

class snorkel.learning.gen_learning.**GenerativeModel**(*class_prior=False*, *lf_prior=False*, *lf_propensity=False*, *lf_class_propensity=False*, *seed=271828*, *name=None*)

A generative model for data programming for binary classification.

Supports dependencies among labeling functions.

Parameters

- **class_prior** – whether to include class label prior factors
- **lf_prior** – whether to include labeling function prior factors
- **lf_propensity** – whether to include labeling function propensity factors
- **lf_class_propensity** – whether to include class-specific labeling function propensity factors
- **seed** – seed for initializing state of Numbskull variables

dep_names = (u'dep_similar', u'dep_fixing', u'dep_reinforcing', u'dep_exclusive')

error_analysis(*session*, *X_test*, *Y_test*, *gold_candidate_set=None*, *b=0.5*, *set_unlabeled_as_neg=True*, *display=True*, *scorer=<class 'snorkel.learning.utils.MentionScorer'>*, ***kwargs*)

Prints full score analysis using the Scorer class, and then returns the a tuple of sets conatining the test candidates bucketed for error analysis, i.e.:

- For binary: TP, FP, TN, FN
- For categorical: correct, incorrect

Parameters

- **X_test** – The input test candidates, as a list or annotation matrix
- **Y_test** – The input test labels, as a list or annotation matrix
- **gold_candidate_set** – Full set of TPs in the test set
- **b** – Decision boundary *for binary setting only*
- **set_unlabeled_as_neg** – Whether to map 0 labels -> -1, *binary setting*
- **display** – Print score report
- **scorer** – The Scorer sub-class to use

learned_lf_stats()

Provides a summary of what the model has learned about the labeling functions. For each labeling function, estimates of the following are provided:

Abstain Accuracy Coverage

[Following are only available for binary tasks] True Positive (TP) False Positive (FP) True Negative (TN) False Negative (FN)

For scoped categoricals, the information provided is for the maximum observed cardinality of any single data point.

WARNING: This uses Gibbs sampling to estimate these values. This will tend to mix poorly when there are many very accurate labeling functions. In this case, this function will assume that the classes are approximately balanced.

load(*model_name=None*, *save_dir=u'checkpoints'*, *verbose=True*)

Load model.

marginals(*L*, *candidate_ranges=None*, *batch_size=None*)

Given an M x N label matrix, returns marginal probabilities for each candidate, depending on classification setting:

- **Binary: Returns M-dim array representing the marginal probability** of each candidate being True
- **Categorical (cardinality = K): Returns M x K dense matrix** representing the marginal probabilities of each candidate being each class.
- **Scoped Categorical (cardinality = K, cardinality_ranges not None):** Returns an M x K *sparse* matrix of marginals.

In the categorical setting, the K values (columns in the marginals matrix) correspond to indices of the Candidate values defined.

next()

optional_names = (u'lf_prior', u'lf_propensity', u'lf_class_propensity')

predictions(*X*, *b=0.5*, *batch_size=None*)
Return numpy array of elements in {-1,0,1} based on predicted marginal probabilities.

representation = False

save(*model_name=None*, *save_dir=u'checkpoints'*, *verbose=True*)
Save current model.

save_marginals(*session*, *X*, *training=False*)
Save the predicted marginal probabilities for the Candidates X.

score(*X_test*, *Y_test*, *b=0.5*, *set_unlabeled_as_neg=True*, *beta=1*, *batch_size=None*)

Returns the summary scores:

- For binary: precision, recall, F-beta score
- For categorical: accuracy

Parameters

- **X_test** – The input test candidates, as a list or annotation matrix
- **Y_test** – The input test labels, as a list or annotation matrix
- **b** – Decision boundary *for binary setting only*
- **set_unlabeled_as_neg** – Whether to map 0 labels -> -1, *binary setting.*
- **beta** – For F-beta score; by default beta = 1 => F-1 score.

Note: Unlike in self.error_analysis, this method assumes X_test and Y_test are properly collated!

train(*L*, *deps=()*, *LF_acc_prior_weights=None*, *LF_acc_prior_weight_default=1*, *labels=None*, *label_prior_weight=5*, *init_deps=0.0*, *init_class_prior=-1.0*, *epochs=30*, *step_size=None*, *decay=1.0*, *reg_param=0.1*, *reg_type=2*, *verbose=False*, *truncation=10*, *burn_in=5*, *cardinality=None*, *timer=None*, *candidate_ranges=None*, *threads=1*)
Fits the parameters of the model to a data set. By default, learns a conditionally independent model. Additional unary dependencies can be set to be included in the constructor. Additional pairwise and higher-order dependencies can be included as an argument.

Results are stored as a member named weights, instance of snorkel.learning.gen_learning.GenerativeModelWeights.

Parameters

- **L** – M x N csr_AnnotationMatrix-type label matrix, where there are M candidates labeled by N labeling functions (LFs)
- **deps** – collection of dependencies to include in the model, each element is a tuple of the form (LF 1 index, LF 2 index, dependency type), see snorkel.learning.constants
- **LF_acc_prior_weights** – An N-element list of prior weights for the LF accuracies (log scale)
- **LF_acc_prior_weight_default** – Default prior for the weight of each LF accuracy; if LF_acc_prior_weights is unset, each LF will have this accuracy prior weight (log scale)
- **labels** – Optional ground truth labels
- **label_prior_weight** – The prior probability that the ground truth labels (if provided) are correct (log scale)
- **init_deps** – initial weight for additional dependencies, except class prior (log scale)

- **init_class_prior** – initial class prior (in log scale), note only used if class_prior=True in constructor
- **epochs** – number of training epochs
- **step_size** – gradient step size, default is 1 / L.shape[0]
- **decay** – multiplicative decay of step size, step_size_(t+1) = step_size_(t) * decay
- **reg_param** – regularization strength
- **reg_type** – 1 = L1 regularization, 2 = L2 regularization
- **verbose** – whether to write debugging info to stdout
- **truncation** – number of iterations between truncation step for L1 regularization
- **burn_in** – number of burn-in samples to take before beginning learning
- **cardinality** – number of possible classes; by default is inferred from the label matrix L
- **timer** – stopwatch for profiling, must implement start() and end()
- **candidate_ranges** – Optionally, a list of M sets of integer values, representing the possible categorical values that each of the M candidates can take. If a label is outside of this range throws an error. If None, then each candidate can take any value from 0 to cardinality.
- **threads** – the number of threads to use for sampling. Default is 1.

class snorkel.learning.gen_learning.**GenerativeModelWeights**(*n*)

is_sign_sparsistent(*other*, *threshold=0.1*)

next()

4.3 Discriminative Models

class snorkel.learning.disc_learning.**TFNoiseAwareModel**(*n_threads=None*, *seed=123*, *deterministic=False*, ***kwargs*)

Generic NoiseAwareModel class for TensorFlow models. Note that the actual network is built when train is called (to allow for model architectures which depend on the training data, e.g. vocab size).

Parameters

- **n_threads** – Parallelism to use; single-threaded if None
- **seed** – Top level seed which is passed into both numpy operations via a RandomState maintained by the class, and into TF as a graph-level seed.

:param deterministic [EXPERIMENTAL / in development!] If True, attempts to make the model deterministic on GPU by replacing all reduce_ and other non-deterministic operations; has no effect (other than potential slight slowdown) for CPU (at least for single-threaded?).

error_analysis(*session*, *X_test*, *Y_test*, *gold_candidate_set=None*, *b=0.5*, *set_unlabeled_as_neg=True*, *display=True*, *scorer=<class 'snorkel.learning.utils.MentionScorer'>*, ***kwargs*)

Prints full score analysis using the Scorer class, and then returns the a tuple of sets conatining the test candidates bucketed for error analysis, i.e.:

- For binary: TP, FP, TN, FN
- For categorical: correct, incorrect

Parameters

- **X_test** – The input test candidates, as a list or annotation matrix
- **Y_test** – The input test labels, as a list or annotation matrix
- **gold_candidate_set** – Full set of TPs in the test set
- **b** – Decision boundary *for binary setting only*
- **set_unlabeled_as_neg** – Whether to map 0 labels -> -1, *binary setting*
- **display** – Print score report
- **scorer** – The Scorer sub-class to use

load(*model_name=None*, *save_dir=u'checkpoints'*, *verbose=True*)
Load model from file and rebuild in new graph / session.

marginals(*X*, *batch_size=None*)
Compute the marginals for the given candidates X. Split into batches to avoid OOM errors, then call _marginals_batch; defaults to no batching.

next()

predictions(*X*, *b=0.5*, *batch_size=None*)
Return numpy array of elements in {-1,0,1} based on predicted marginal probabilities.

representation = False

save(*model_name=None*, *save_dir=u'checkpoints'*, *verbose=True*, *global_step=0*)
Save current model.

save_marginals(*session*, *X*, *training=False*)
Save the predicted marginal probabilities for the Candidates X.

score(*X_test*, *Y_test*, *b=0.5*, *set_unlabeled_as_neg=True*, *beta=1*, *batch_size=None*)

Returns the summary scores:

- For binary: precision, recall, F-beta score
- For categorical: accuracy

Parameters

- **X_test** – The input test candidates, as a list or annotation matrix
- **Y_test** – The input test labels, as a list or annotation matrix
- **b** – Decision boundary *for binary setting only*
- **set_unlabeled_as_neg** – Whether to map 0 labels -> -1, *binary setting.*
- **beta** – For F-beta score; by default beta = 1 => F-1 score.

Note: Unlike in self.error_analysis, this method assumes X_test and Y_test are properly collated!

train(*X_train*, *Y_train*, *n_epochs=25*, *lr=0.01*, *batch_size=256*, *rebalance=False*, *X_dev=None*, *Y_dev=None*, *print_freq=5*, *dev_ckpt=True*, *dev_ckpt_delay=0.75*, *save_dir=u'checkpoints'*, ***kwargs*)
Generic training procedure for TF model

Parameters

- **X_train** – The training Candidates. If self.representation is True, then this is a list of Candidate objects; else is a csr_AnnotationMatrix with rows corresponding to training candidates and columns corresponding to features.
- **Y_train** – Array of marginal probabilities for each Candidate
- **n_epochs** – Number of training epochs
- **lr** – Learning rate
- **batch_size** – Batch size for SGD
- **rebalance** – Bool or fraction of positive examples for training - if True, defaults to standard 0.5 class balance - if False, no class balancing
- **X_dev** – Candidates for evaluation, same format as X_train
- **Y_dev** – Labels for evaluation, same format as Y_train
- **print_freq** – number of epochs at which to print status, and if present, evaluate the dev set (X_dev, Y_dev).
- **dev_ckpt** – If True, save a checkpoint whenever highest score on (X_dev, Y_dev) reached. Note: currently only evaluates at every @print_freq epochs.
- **dev_ckpt_delay** – Start dev checkpointing after this portion of n_epochs.
- **save_dir** – Save dir path for checkpointing.
- **kwargs** – All hyperparameters that change how the graph is built must be passed through here to be saved and reloaded to save / reload model. *NOTE: If a parameter needed to build the network and/or is needed at test time is not included here, the model will not be able to be reloaded!*

class snorkel.learning.disc_models.logistic_regression.**LogisticRegression**(*n_threads=None*, *seed=123*, *deterministic=False*, ***kwargs*)

error_analysis(*session*, *X_test*, *Y_test*, *gold_candidate_set=None*, *b=0.5*, *set_unlabeled_as_neg=True*, *display=True*, *scorer=<class 'snorkel.learning.utils.MentionScorer'>*, ***kwargs*)

Prints full score analysis using the Scorer class, and then returns the a tuple of sets conatining the test candidates bucketed for error analysis, i.e.:

- For binary: TP, FP, TN, FN
- For categorical: correct, incorrect

Parameters

- **X_test** – The input test candidates, as a list or annotation matrix
- **Y_test** – The input test labels, as a list or annotation matrix
- **gold_candidate_set** – Full set of TPs in the test set
- **b** – Decision boundary *for binary setting only*

- **set_unlabeled_as_neg** – Whether to map 0 labels -> -1, *binary setting*
- **display** – Print score report
- **scorer** – The Scorer sub-class to use

get_weights()
Get model weights and bias

load(*model_name=None*, *save_dir=u'checkpoints'*, *verbose=True*)
Load model from file and rebuild in new graph / session.

marginals(*X*, *batch_size=None*)
Compute the marginals for the given candidates X. Split into batches to avoid OOM errors, then call _marginals_batch; defaults to no batching.

next()

predictions(*X*, *b=0.5*, *batch_size=None*)
Return numpy array of elements in {-1,0,1} based on predicted marginal probabilities.

representation = False

save(*model_name=None*, *save_dir=u'checkpoints'*, *verbose=True*, *global_step=0*)
Save current model.

save_marginals(*session*, *X*, *training=False*)
Save the predicted marginal probabilities for the Candidates X.

score(*X_test*, *Y_test*, *b=0.5*, *set_unlabeled_as_neg=True*, *beta=1*, *batch_size=None*)

Returns the summary scores:

- For binary: precision, recall, F-beta score
- For categorical: accuracy

Parameters

- **X_test** – The input test candidates, as a list or annotation matrix
- **Y_test** – The input test labels, as a list or annotation matrix
- **b** – Decision boundary *for binary setting only*
- **set_unlabeled_as_neg** – Whether to map 0 labels -> -1, *binary setting.*
- **beta** – For F-beta score; by default beta = 1 => F-1 score.

Note: Unlike in self.error_analysis, this method assumes X_test and Y_test are properly collated!

train(*X_train*, *Y_train*, *n_epochs=25*, *lr=0.01*, *batch_size=256*, *rebalance=False*, *X_dev=None*, *Y_dev=None*, *print_freq=5*, *dev_ckpt=True*, *dev_ckpt_delay=0.75*, *save_dir=u'checkpoints'*, ***kwargs*)
Generic training procedure for TF model

Parameters

- **X_train** – The training Candidates. If self.representation is True, then this is a list of Candidate objects; else is a csr_AnnotationMatrix with rows corresponding to training candidates and columns corresponding to features.
- **Y_train** – Array of marginal probabilities for each Candidate
- **n_epochs** – Number of training epochs

- **lr** – Learning rate
- **batch_size** – Batch size for SGD
- **rebalance** – Bool or fraction of positive examples for training - if True, defaults to standard 0.5 class balance - if False, no class balancing
- **X_dev** – Candidates for evaluation, same format as X_train
- **Y_dev** – Labels for evaluation, same format as Y_train
- **print_freq** – number of epochs at which to print status, and if present, evaluate the dev set (X_dev, Y_dev).
- **dev_ckpt** – If True, save a checkpoint whenever highest score on (X_dev, Y_dev) reached. Note: currently only evaluates at every @print_freq epochs.
- **dev_ckpt_delay** – Start dev checkpointing after this portion of n_epochs.
- **save_dir** – Save dir path for checkpointing.
- **kwargs** – All hyperparameters that change how the graph is built must be passed through here to be saved and reloaded to save / reload model. *NOTE: If a parameter needed to build the network and/or is needed at test time is not included here, the model will not be able to be reloaded!*

class snorkel.learning.disc_models.logistic_regression.**SparseLogisticRegression**(*n_threads=Non seed=123, deterministic=False, **kwargs*)

error_analysis(*session*, *X_test*, *Y_test*, *gold_candidate_set=None*, *b=0.5*, *set_unlabeled_as_neg=True*, *display=True*, *scorer=<class 'snorkel.learning.utils.MentionScorer'>*, ***kwargs*)

Prints full score analysis using the Scorer class, and then returns the a tuple of sets conatining the test candidates bucketed for error analysis, i.e.:

- For binary: TP, FP, TN, FN
- For categorical: correct, incorrect

Parameters

- **X_test** – The input test candidates, as a list or annotation matrix
- **Y_test** – The input test labels, as a list or annotation matrix
- **gold_candidate_set** – Full set of TPs in the test set
- **b** – Decision boundary *for binary setting only*
- **set_unlabeled_as_neg** – Whether to map 0 labels -> -1, *binary setting*
- **display** – Print score report
- **scorer** – The Scorer sub-class to use

get_weights()

Get model weights and bias

load(*model_name=None*, *save_dir=u'checkpoints'*, *verbose=True*)
Load model from file and rebuild in new graph / session.

marginals(*X*, *batch_size=None*)
Compute the marginals for the given candidates X. Split into batches to avoid OOM errors, then call _marginals_batch; defaults to no batching.

next()

predictions(*X*, *b=0.5*, *batch_size=None*)
Return numpy array of elements in {-1,0,1} based on predicted marginal probabilities.

representation = False

save(*model_name=None*, *save_dir=u'checkpoints'*, *verbose=True*, *global_step=0*)
Save current model.

save_marginals(*session*, *X*, *training=False*)
Save the predicted marginal probabilities for the Candidates X.

score(*X_test*, *Y_test*, *b=0.5*, *set_unlabeled_as_neg=True*, *beta=1*, *batch_size=None*)

Returns the summary scores:

- For binary: precision, recall, F-beta score
- For categorical: accuracy

Parameters

- **X_test** – The input test candidates, as a list or annotation matrix
- **Y_test** – The input test labels, as a list or annotation matrix
- **b** – Decision boundary *for binary setting only*
- **set_unlabeled_as_neg** – Whether to map 0 labels -> -1, *binary setting.*
- **beta** – For F-beta score; by default beta = 1 => F-1 score.

Note: Unlike in self.error_analysis, this method assumes X_test and Y_test are properly collated!

train(*X_train*, *Y_train*, *n_epochs=25*, *lr=0.01*, *batch_size=256*, *rebalance=False*, *X_dev=None*, *Y_dev=None*, *print_freq=5*, *dev_ckpt=True*, *dev_ckpt_delay=0.75*, *save_dir=u'checkpoints'*, ***kwargs*)
Generic training procedure for TF model

Parameters

- **X_train** – The training Candidates. If self.representation is True, then this is a list of Candidate objects; else is a csr_AnnotationMatrix with rows corresponding to training candidates and columns corresponding to features.
- **Y_train** – Array of marginal probabilities for each Candidate
- **n_epochs** – Number of training epochs
- **lr** – Learning rate
- **batch_size** – Batch size for SGD
- **rebalance** – Bool or fraction of positive examples for training - if True, defaults to standard 0.5 class balance - if False, no class balancing
- **X_dev** – Candidates for evaluation, same format as X_train
- **Y_dev** – Labels for evaluation, same format as Y_train

- **print_freq** – number of epochs at which to print status, and if present, evaluate the dev set (X_dev, Y_dev).
- **dev_ckpt** – If True, save a checkpoint whenever highest score on (X_dev, Y_dev) reached. Note: currently only evaluates at every @print_freq epochs.
- **dev_ckpt_delay** – Start dev checkpointing after this portion of n_epochs.
- **save_dir** – Save dir path for checkpointing.
- **kwargs** – All hyperparameters that change how the graph is built must be passed through here to be saved and reloaded to save / reload model. *NOTE: If a parameter needed to build the network and/or is needed at test time is not included here, the model will not be able to be reloaded!*

class snorkel.learning.disc_models.rnn.rnn_base.**RNNBase**(*n_threads=None*, *seed=123*, *deterministic=False*, ***kwargs*)

error_analysis(*session*, *X_test*, *Y_test*, *gold_candidate_set=None*, *b=0.5*, *set_unlabeled_as_neg=True*, *display=True*, *scorer=<class 'snorkel.learning.utils.MentionScorer'>*, ***kwargs*)

Prints full score analysis using the Scorer class, and then returns the a tuple of sets conatining the test candidates bucketed for error analysis, i.e.:

- For binary: TP, FP, TN, FN
- For categorical: correct, incorrect

Parameters

- **X_test** – The input test candidates, as a list or annotation matrix
- **Y_test** – The input test labels, as a list or annotation matrix
- **gold_candidate_set** – Full set of TPs in the test set
- **b** – Decision boundary *for binary setting only*
- **set_unlabeled_as_neg** – Whether to map 0 labels -> -1, *binary setting*
- **display** – Print score report
- **scorer** – The Scorer sub-class to use

load(*model_name=None*, *save_dir=u'checkpoints'*, *verbose=True*)

Load model from file and rebuild in new graph / session.

marginals(*X*, *batch_size=None*)

Compute the marginals for the given candidates X. Split into batches to avoid OOM errors, then call _marginals_batch; defaults to no batching.

next()

predictions(*X*, *b=0.5*, *batch_size=None*)

Return numpy array of elements in {-1,0,1} based on predicted marginal probabilities.

representation = True

save(*model_name=None*, *save_dir=u'checkpoints'*, *verbose=True*, *global_step=0*)

Save current model.

save_marginals(*session*, *X*, *training=False*)

Save the predicted marginal probabilities for the Candidates X.

score(*X_test*, *Y_test*, *b=0.5*, *set_unlabeled_as_neg=True*, *beta=1*, *batch_size=None*)

Returns the summary scores:

- For binary: precision, recall, F-beta score
- For categorical: accuracy

Parameters

- **X_test** – The input test candidates, as a list or annotation matrix
- **Y_test** – The input test labels, as a list or annotation matrix
- **b** – Decision boundary *for binary setting only*
- **set_unlabeled_as_neg** – Whether to map 0 labels -> -1, *binary setting.*
- **beta** – For F-beta score; by default beta = 1 => F-1 score.

Note: Unlike in self.error_analysis, this method assumes X_test and Y_test are properly collated!

train(*X_train*, *Y_train*, *X_dev=None*, *max_sentence_length=None*, ***kwargs*)
Perform preprocessing of data, construct dataset-specific model, then train.

snorkel.learning.disc_models.rnn.re_rnn.**mark**(*l*, *h*, *idx*)
Produce markers based on argument positions

Parameters

- **l** – sentence position of first word in argument
- **h** – sentence position of last word in argument
- **idx** – argument index (1 or 2)

snorkel.learning.disc_models.rnn.re_rnn.**mark_sentence**(*s*, *args*)
Insert markers around relation arguments in word sequence

Parameters

- **s** – list of tokens in sentence
- **args** – list of triples (l, h, idx) as per @_mark(...) corresponding to relation arguments

Example: Then Barack married Michelle. -> Then ~~[[1 Barack 1]]~~ married ~~[[2 Michelle 2]]~~.

class snorkel.learning.disc_models.rnn.re_rnn.**reRNN**(*n_threads=None*, *seed=123*, *deterministic=False*, ***kwargs*)
reRNN for relation extraction

error_analysis(*session*, *X_test*, *Y_test*, *gold_candidate_set=None*, *b=0.5*, *set_unlabeled_as_neg=True*, *display=True*, *scorer=<class 'snorkel.learning.utils.MentionScorer'>*, ***kwargs*)
Prints full score analysis using the Scorer class, and then returns the a tuple of sets conatining the test candidates bucketed for error analysis, i.e.:

- For binary: TP, FP, TN, FN
- For categorical: correct, incorrect

Parameters

- **X_test** – The input test candidates, as a list or annotation matrix
- **Y_test** – The input test labels, as a list or annotation matrix

- **gold_candidate_set** – Full set of TPs in the test set
- **b** – Decision boundary *for binary setting only*
- **set_unlabeled_as_neg** – Whether to map 0 labels -> -1, *binary setting*
- **display** – Print score report
- **scorer** – The Scorer sub-class to use

load(*model_name=None*, *save_dir=u'checkpoints'*, *verbose=True*)
Load model from file and rebuild in new graph / session.

marginals(*X*, *batch_size=None*)
Compute the marginals for the given candidates X. Split into batches to avoid OOM errors, then call _marginals_batch; defaults to no batching.

next()

predictions(*X*, *b=0.5*, *batch_size=None*)
Return numpy array of elements in {-1,0,1} based on predicted marginal probabilities.

representation = True

save(*model_name=None*, *save_dir=u'checkpoints'*, *verbose=True*, *global_step=0*)
Save current model.

save_marginals(*session*, *X*, *training=False*)
Save the predicted marginal probabilities for the Candidates X.

score(*X_test*, *Y_test*, *b=0.5*, *set_unlabeled_as_neg=True*, *beta=1*, *batch_size=None*)

Returns the summary scores:

- For binary: precision, recall, F-beta score
- For categorical: accuracy

Parameters

- **X_test** – The input test candidates, as a list or annotation matrix
- **Y_test** – The input test labels, as a list or annotation matrix
- **b** – Decision boundary *for binary setting only*
- **set_unlabeled_as_neg** – Whether to map 0 labels -> -1, *binary setting.*
- **beta** – For F-beta score; by default beta = 1 => F-1 score.

Note: Unlike in self.error_analysis, this method assumes X_test and Y_test are properly collated!

train(*X_train*, *Y_train*, *X_dev=None*, *max_sentence_length=None*, ***kwargs*)
Perform preprocessing of data, construct dataset-specific model, then train.

class snorkel.learning.disc_models.rnn.tag_rnn.**TagRNN**(*n_threads=None*, *seed=123*, *deterministic=False*, ***kwargs*)

TagRNN for sequence tagging

CLOSE = u'~~]]~~'

OPEN = u'~~[[~~'

error_analysis(*session*, *X_test*, *Y_test*, *gold_candidate_set=None*, *b=0.5*, *set_unlabeled_as_neg=True*, *display=True*, *scorer=<class 'snorkel.learning.utils.MentionScorer'>*, ***kwargs*)

Prints full score analysis using the Scorer class, and then returns the a tuple of sets conatining the test candidates bucketed for error analysis, i.e.:

- For binary: TP, FP, TN, FN
- For categorical: correct, incorrect

Parameters

- **X_test** – The input test candidates, as a list or annotation matrix
- **Y_test** – The input test labels, as a list or annotation matrix
- **gold_candidate_set** – Full set of TPs in the test set
- **b** – Decision boundary *for binary setting only*
- **set_unlabeled_as_neg** – Whether to map 0 labels -> -1, *binary setting*
- **display** – Print score report
- **scorer** – The Scorer sub-class to use

load(*model_name=None*, *save_dir=u'checkpoints'*, *verbose=True*)

Load model from file and rebuild in new graph / session.

marginals(*X*, *batch_size=None*)

Compute the marginals for the given candidates X. Split into batches to avoid OOM errors, then call _marginals_batch; defaults to no batching.

next()

predictions(*X*, *b=0.5*, *batch_size=None*)

Return numpy array of elements in {-1,0,1} based on predicted marginal probabilities.

representation = True

save(*model_name=None*, *save_dir=u'checkpoints'*, *verbose=True*, *global_step=0*)

Save current model.

save_marginals(*session*, *X*, *training=False*)

Save the predicted marginal probabilities for the Candidates X.

score(*X_test*, *Y_test*, *b=0.5*, *set_unlabeled_as_neg=True*, *beta=1*, *batch_size=None*)

Returns the summary scores:

- For binary: precision, recall, F-beta score
- For categorical: accuracy

Parameters

- **X_test** – The input test candidates, as a list or annotation matrix
- **Y_test** – The input test labels, as a list or annotation matrix
- **b** – Decision boundary *for binary setting only*
- **set_unlabeled_as_neg** – Whether to map 0 labels -> -1, *binary setting.*
- **beta** – For F-beta score; by default beta = 1 => F-1 score.

Note: Unlike in self.error_analysis, this method assumes X_test and Y_test are properly collated!

train(*X_train*, *Y_train*, *X_dev=None*, *max_sentence_length=None*, ***kwargs*)
Perform preprocessing of data, construct dataset-specific model, then train.

snorkel.learning.disc_models.rnn.tag_rnn.**tag**(*seq*, *labels*)

class snorkel.learning.disc_models.rnn.text_rnn.**TextRNN**(*n_threads=None*, *seed=123*, *deterministic=False*, ***kwargs*)

TextRNN for strings of text.

error_analysis(*session*, *X_test*, *Y_test*, *gold_candidate_set=None*, *b=0.5*, *set_unlabeled_as_neg=True*, *display=True*, *scorer=<class 'snorkel.learning.utils.MentionScorer'>*, ***kwargs*)
Prints full score analysis using the Scorer class, and then returns the a tuple of sets conatining the test candidates bucketed for error analysis, i.e.:

- For binary: TP, FP, TN, FN
- For categorical: correct, incorrect

Parameters

- **X_test** – The input test candidates, as a list or annotation matrix
- **Y_test** – The input test labels, as a list or annotation matrix
- **gold_candidate_set** – Full set of TPs in the test set
- **b** – Decision boundary *for binary setting only*
- **set_unlabeled_as_neg** – Whether to map 0 labels -> -1, *binary setting*
- **display** – Print score report
- **scorer** – The Scorer sub-class to use

load(*model_name=None*, *save_dir=u'checkpoints'*, *verbose=True*)
Load model from file and rebuild in new graph / session.

marginals(*X*, *batch_size=None*)
Compute the marginals for the given candidates X. Split into batches to avoid OOM errors, then call _marginals_batch; defaults to no batching.

next()

predictions(*X*, *b=0.5*, *batch_size=None*)
Return numpy array of elements in {-1,0,1} based on predicted marginal probabilities.

representation = True

save(*model_name=None*, *save_dir=u'checkpoints'*, *verbose=True*, *global_step=0*)
Save current model.

save_marginals(*session*, *X*, *training=False*)
Save the predicted marginal probabilities for the Candidates X.

score(*X_test*, *Y_test*, *b=0.5*, *set_unlabeled_as_neg=True*, *beta=1*, *batch_size=None*)

Returns the summary scores:

- For binary: precision, recall, F-beta score
- For categorical: accuracy

Parameters

- **X_test** – The input test candidates, as a list or annotation matrix
- **Y_test** – The input test labels, as a list or annotation matrix
- **b** – Decision boundary *for binary setting only*
- **set_unlabeled_as_neg** – Whether to map 0 labels -> -1, *binary setting.*
- **beta** – For F-beta score; by default beta = 1 => F-1 score.

Note: Unlike in self.error_analysis, this method assumes X_test and Y_test are properly collated!

train(*X_train*, *Y_train*, *X_dev=None*, *max_sentence_length=None*, ***kwargs*)
Perform preprocessing of data, construct dataset-specific model, then train.

4.4 Learning Utilities

class snorkel.learning.utils.**GridSearch**(*model_class*, *parameter_dict*, *X_train*, *Y_train=None*, *model_class_params={}*, *model_hyperparams={}*, *save_dir=u'checkpoints'*)
A class for running a hyperparameter grid search.

Parameters

- **model_class** – The model class being trained
- **parameter_dict** – A dictionary of (hyperparameter name, list of values) pairs. Note that the hyperparameter name must correspond to a keyword argument in the *model_class.train* method.
- **X_train** – The training datapoints
- **Y_train** – If applicable, the training labels / marginals
- **model_class_params** – Keyword arguments to pass into model_class construction. Note that a new model is constructed for each new combination of hyperparameters.
- **model_hyperparams** – Hyperparameters for the model- all must be keyword arguments to the *model_class.train* method. Any that are included in the grid search will be overwritten.
- **save_dir** – Note that checkpoints will be saved in save_dir/grid_search

fit(*X_valid*, *Y_valid*, *b=0.5*, *beta=1*, *set_unlabeled_as_neg=True*, *n_threads=1*, *eval_batch_size=None*)
Runs grid search, constructing a new instance of model_class for each hyperparameter combination, training on (self.X_train, self.Y_train), and validating on (X_valid, Y_valid). Selects the best model according to F1 score (binary) or accuracy (categorical).

Parameters

- **b** – Scoring decision threshold (binary)
- **beta** – F_beta score to select model by (binary)
- **set_unlabeled_as_neg** – Set labels = 0 -> -1 (binary)
- **n_threads** – Parallelism to use for the grid search
- **eval_batch_size** – The batch_size for model evaluation

search_space()

snorkel.learning.utils.**LF_accuracies**(*L*, *labels*)
Given an N x M matrix where L_{i,j} is the label given by the jth LF to the ith candidate, and labels {-1,1} Return the accuracy of each LF w.r.t. these labels

snorkel.learning.utils.**LF_conflicts**(*L*)
Given an N x M matrix where L_{i,j} is the label given by the jth LF to the ith candidate: Return the **fraction of candidates that each LF _conflicts with other LFs on_.**

snorkel.learning.utils.**LF_coverage**(*L*)
Given an N x M matrix where L_{i,j} is the label given by the jth LF to the ith candidate: Return the **fraction of candidates that each LF labels.**

snorkel.learning.utils.**LF_overlaps**(*L*)
Given an N x M matrix where L_{i,j} is the label given by the jth LF to the ith candidate: Return the **fraction of candidates that each LF _overlaps with other LFs on_.**

class snorkel.learning.utils.**LabelBalancer**(*y*)

get_train_idxs(*rebalance=False*, *split=0.5*, *rand_state=None*)
Get training indices based on @y @rebalance: bool or fraction of positive examples desired

If True, default fraction is 0.5. If False no balancing.

@split: Split point for positive and negative classes

class snorkel.learning.utils.**MentionScorer**(*test_candidates*, *test_labels*, *gold_candidate_set=None*)
Scorer for mention level assessment

score(*test_marginals*, ***kwargs*)

summary_score(*test_marginals*, ***kwargs*)
Return the F1 score (for binary) or accuracy (for categorical). Also return the label as second argument.

class snorkel.learning.utils.**ModelTester**(*model_class*, *model_class_params*, *params_queue*, *scores_queue*, *X_train*, *X_valid*, *Y_valid*, *Y_train=None*, *b=0.5*, *beta=1*, *set_unlabeled_as_neg=True*, *save_dir=u'checkpoints'*, *eval_batch_size=None*)

authkey

daemon
Return whether process is a daemon

exitcode
Return exit code of process or *None* if it has yet to stop

ident
Return identifier (PID) of process or *None* if it has yet to start

is_alive()
Return whether process is alive

join(*timeout=None*)
Wait until child process terminates

name

pid
Return identifier (PID) of process or *None* if it has yet to start

run()

start()
Start child process

terminate()
Terminate process; sends SIGTERM signal or uses TerminateProcess()

class snorkel.learning.utils.**RandomSearch**(*model_class*, *parameter_dict*, *X_train*, *Y_train=None*, *n=10*, *model_class_params={}*, *model_hyperparams={}*, *seed=123*, *save_dir=u'checkpoints'*)
A GridSearch over a random subsample of the hyperparameter search space.

Parameters **seed** – A seed for the GridSearch instance

fit(*X_valid*, *Y_valid*, *b=0.5*, *beta=1*, *set_unlabeled_as_neg=True*, *n_threads=1*, *eval_batch_size=None*)
Runs grid search, constructing a new instance of model_class for each hyperparameter combination, training on (self.X_train, self.Y_train), and validating on (X_valid, Y_valid). Selects the best model according to F1 score (binary) or accuracy (categorical).

Parameters

- **b** – Scoring decision threshold (binary)
- **beta** – F_beta score to select model by (binary)
- **set_unlabeled_as_neg** – Set labels = 0 -> -1 (binary)
- **n_threads** – Parallelism to use for the grid search
- **eval_batch_size** – The batch_size for model evaluation

search_space()

class snorkel.learning.utils.**Scorer**(*test_candidates*, *test_labels*, *gold_candidate_set=None*)
Abstract type for scorers

score(*test_marginals*, ***kwargs*)

summary_score(*test_marginals*, ***kwargs*)
Return the F1 score (for binary) or accuracy (for categorical).

snorkel.learning.utils.**binary_scores_from_counts**(*ntp*, *nfp*, *ntn*, *nfn*)
Precision, recall, and F1 scores from counts of TP, FP, TN, FN. Example usage:

p, r, f1 = binary_scores_from_counts(*map(len, error_sets))

snorkel.learning.utils.**candidate_conflict**(*L*)
Given an N x M matrix where L_{i,j} is the label given by the jth LF to the ith candidate: Return the **fraction of candidates which have > 1 (non-zero) labels _which are not equal_.**

snorkel.learning.utils.**candidate_coverage**(*L*)
Given an N x M matrix where L_{i,j} is the label given by the jth LF to the ith candidate: Return the **fraction of candidates which have > 0 (non-zero) labels.**

snorkel.learning.utils.**candidate_overlap**(*L*)
Given an N x M matrix where L_{i,j} is the label given by the jth LF to the ith candidate: Return the **fraction of candidates which have > 1 (non-zero) labels.**

snorkel.learning.utils.**print_scores**(*ntp*, *nfp*, *ntn*, *nfn*, *title=u'Scores'*)

snorkel.learning.utils.**reshape_marginals**(*marginals*)
Returns correctly shaped marginals as np array

snorkel.learning.utils.**sparse_abs**(*X*)

Element-wise absolute value of sparse matrix- avoids casting to dense matrix!

snorkel.learning.utils.**training_set_summary_stats**(*L*, *return_vals=True*, *verbose=False*)

Given an N x M matrix where L_{i,j} is the label given by the jth LF to the ith candidate: Return simple summary statistics

CHAPTER 5

Etc: Viewing and Annotating Data, Writing LFs

5.1 Using the Viewer to Inspect and Annotate Data

class snorkel.viewer.**SentenceNgramViewer**(*candidates*, *session*, *gold=[]*, *n_per_page=3*, *height=225*, *annotator_name=None*)

Viewer for Sentence objects and candidate Spans within them

class snorkel.viewer.**Viewer**(*candidates*, *session*, *gold=[]*, *n_per_page=3*, *height=225*, *annotator_name=None*)

Generic object for viewing and labeling Candidate objects in their rendered Contexts.

handle_label_event(*_*, *content*, *buffers*)

Handles label event by persisting new label

render()

Renders viewer pane

5.2 Helpers for Writing Labeling Functions

snorkel.lf_helpers.**contains_token**(*c*, *tok*, *attrib=u'words'*, *case_sensitive=False*)

Checks if any of the contituent Spans contain a token :param attrib: The token attribute type (e.g. words, lemmas, poses)

snorkel.lf_helpers.**get_between_tokens**(*c*, *attrib=u'words'*, *n_max=1*, *case_sensitive=False*)

TODO: write doc_string

snorkel.lf_helpers.**get_doc_candidate_spans**(*c*)

Get the Spans in the same document as Candidate c, where these Spans are arguments of Candidates.

snorkel.lf_helpers.**get_left_tokens**(*c*, *window=3*, *attrib=u'words'*, *n_max=1*, *case_sensitive=False*)

Return the tokens within a window to the _left_ of the Candidate. For higher-arity Candidates, defaults to the _first_ argument. :param window: The number of tokens to the left of the first argument to

return

Parameters **attrib** – The token attribute type (e.g. words, lemmas, poses)

snorkel.lf_helpers.**get_matches**(*lf, candidate_set, match_values=[1, -1]*)
A simple helper function to see how many matches (non-zero by default) an LF gets. Returns the matched set, which can then be directly put into the Viewer.

snorkel.lf_helpers.**get_right_tokens**(*c*, *window=3*, *attrib=u'words'*, *n_max=1*, *case_sensitive=False*)
Return the tokens within a window to the _right_ of the Candidate. For higher-arity Candidates, defaults to the _last_ argument. :param window: The number of tokens to the right of the last argument to

return

Parameters **attrib** – The token attribute type (e.g. words, lemmas, poses)

snorkel.lf_helpers.**get_sent_candidate_spans**(*c*)
Get the Spans in the same Sentence as Candidate c, where these Spans are arguments of Candidates.

snorkel.lf_helpers.**get_tagged_text**(*c*)
Returns the text of c's parent context with c's unary spans replaced with tags {{A}}, {{B}}, etc. A convenience method for writing LFs based on e.g. regexes.

snorkel.lf_helpers.**get_text_between**(*c*)
Returns the text between the two unary Spans of a binary-Span Candidate, where both are in the same Sentence.

snorkel.lf_helpers.**get_text_splits**(*c*)
Given a k-arity Candidate defined over k Spans, return the chunked parent context (e.g. Sentence) split around the k constituent Spans.

NOTE: Currently assumes that these Spans are in the same Context

snorkel.lf_helpers.**is_inverted**(*c*)
Returns True if the ordering of the candidates in the sentence is inverted.

snorkel.lf_helpers.**test_LF**(*session*, *lf*, *split*, *annotator_name*)
Gets the accuracy of a single LF on a split of the candidates, w.r.t. annotator labels, and also returns the error buckets of the candidates.

5.3 Helpers for Loading External Annotations

Python Module Index

S

Index

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X