Enso Documentation

Release 0.1

Slater Victoroff, Madison May, Maritza Ebling

Apr 30, 2018

Contents

An overview of the developer interface to Enso.

CHAPTER 1

Enso workflow

Enso is tool intended to provide a standard interface for the benchmarking of embedding and transfer learning methods for natural language processing tasks. Although there are other effective approaches to applying transfer learning to natural language processing, it's built on the assumption that the approach to "transfer learning" adheres to the below flow. This approach is designed to replicate a scenario where a pool of unlabeled data is available, and labelers with subject matter expertise have a limited amount of time to provide labels for a subset of the unlabeled data.

- Download pre-ETL'ed source datasets for testing (*python -m enso.download*)
- All examples in the dataset are "featurized" via a pre-trained source model (*python -m enso.featurize*)
- Re-represented data is separated into train and test sets
- A fixed number of examples from the train set is selected to use as training data via the selected sampling strategy
- The training data subset is optionally over or under-sampled to account for variation in class balance
- A target model is trained using the featurized training examples as inputs (*python -m enso.experiment*)
- The target model is benchmarked on all featurized test examples
- The process is repeated for all combinations of featurizers, dataset sizes, target model architectures, etc.
- Results are visualized and manually inspected (*python -m enso.visualize*)

CHAPTER 2

Running Experiments with Enso

Each component of Enso is designed to be extensible and customizable. Base classes for `enso.Featurizer`, `enso.Sampler`, `enso.Experiment`, `enso.Metric` and `enso.Visualizer` are provided in order to enable anyone to implement and test their own ideas. Subclass those base classes and modify *enso/config.py* to run your own experiments, and consider contributing the results back to the community to help other community members test against better baselines.

CHAPTER 3

Enso Configuration

Experiment settings are managed through the modification of *enso/config.py*. The main parameters of interest are:

- *DATASETS*: A list of the datasets that you want to include in your experiments.
- *FEATURES*: A list of pre-computed features to include in your experiments. Only features for your specified datasets will be used.
- *EXPERIMENTS*: A list of the experiments to run on top of the feature sets that have selected.
- *METRICS*: A list of metrics you'd like to see for the combination of experiments being run
- ***TEST_SETUP*: More detailed test information, likely to vary quite a bit from run to run.**
  - *train_sizes*: A list of training set sizes to be experimented with.
  - *n_splits*: The number of CV splits to perform on each run.
- *VISUALIZATIONS*: A list of the visualizations to create for result visualization.
- ***VISUALIZATION_SETUP*: More detailed visualization information with visualization-specific options.**
  - <*visualization_name*>: Mapping of all the visualization-specific options you want to pass

CHAPTER 4

Dataset Formatting

In order to be a valid dataset, each dataset csv in the *Data* folder must include a *"Text"* column and a *"Target"* column. For now, the "Target" column must be a string class label. No rows may have missing values.

CHAPTER 5

Featurization

5.1 Base Classes

class enso.featurize.**Featurizer**(**args*, ***kwargs*)

Base class for building featurizers.

featurize(*text*)

Parameters **text** – text of a singular example

Returns *np.ndarray* representation of text

featurize_batch(*X*, *batch_size=32*)

Parameters

- **X** – *pd.Series* that contains raw text to featurize
- **batch_size** – int number of examples to process per batch

Returns list of np.ndarray representations of text

generate(*dataset*, *dataset_name*)

Given a *dataset* pandas DataFrame and string *dataset_name*, add column *"Features"* to the provided *pd.DataFrame* and serialize the result to the results folder listed in *config.py*.

If a given featurizer exposes a `featurize_batch()` method, that method will be called to perform featurization. Otherwise, `Featurizer`'s will fall back to calling `featurize()` on each individual example.

Parameters

- **dataset** – *pd.DataFrame* object that must contain a *Text* column.
- **dataset_name** – *str* name to use as a save location in the *config.FEATURES_DIRECTORY*.

load()
Method called in flow of *python -m enso.featurize* to prevent loading pre-trained models into memory on file import.

If loading a pre-trained model into memory is not required, *Featurizer.load()* defaults to *pass*.

name()
Prints the name of the current class to aid logging and result formatting.

5.2 Local Featurizers

These Featurizers will be run on your local machine to embed training and test examples.

5.3 Hosted Featurizers

The organization behind *enso*, indico, hosts a variety of pre-trained models that you can employ as source models for your experiments. These API wrappers assume that an *INDICO_API_KEY* env variable is present in order to authenticate calls made to the indico API. If you would like to test / benchmark indico's hosted embeddings on larger data volumes, reach out to contact@indico.io and inquire about free API credit for academic use.

class enso.featurize.indico_features.**IndicoStandard**(**args*, ***kwargs*)
Featurizer that uses indico's standard features.

featurize(*text*)

Parameters **text** – text of a singular example

Returns *np.ndarray* representation of text

featurize_batch(*X*, *batch_size=32*, ***kwargs*)

Parameters

- **X** – *pd.Series* that contains raw text to featurize
- **batch_size** – int number of examples to process per batch

Returns list of np.ndarray representations of text

generate(*dataset*, *dataset_name*)
Given a *dataset* pandas DataFrame and string *dataset_name*, add column *"Features"* to the provided *pd.DataFrame* and serialize the result to the results folder listed in *config.py*.

If a given featurizer exposes a *featurize_batch()* method, that method will be called to perform featurization. Otherwise, Featurizer's will fall back to calling *featurize()* on each individual example.

Parameters

- **dataset** – *pd.DataFrame* object that must contain a *Text* column.
- **dataset_name** – *str* name to use as a save location in the *config.FEATURES_DIRECTORY*.

load()
Method called in flow of *python -m enso.featurize* to prevent loading pre-trained models into memory on file import.

If loading a pre-trained model into memory is not required, *Featurizer.load()* defaults to *pass*.

name()
Prints the name of the current class to aid logging and result formatting.

class enso.featurize.indico_features.**IndicoSentiment**(**args*, ***kwargs*)
Featurizer that uses indico's sentiment features.

featurize(*text*)

Parameters **text** – text of a singular example

Returns *np.ndarray* representation of text

featurize_batch(*X*, *batch_size=32*, ***kwargs*)

Parameters

- **X** – *pd.Series* that contains raw text to featurize
- **batch_size** – int number of examples to process per batch

Returns list of np.ndarray representations of text

generate(*dataset*, *dataset_name*)
Given a *dataset* pandas DataFrame and string *dataset_name*, add column *"Features"* to the provided *pd.DataFrame* and serialize the result to the results folder listed in *config.py*.

If a given featurizer exposes a *featurize_batch()* method, that method will be called to perform featurization. Otherwise, Featurizer's will fall back to calling *featurize()* on each individual example.

Parameters

- **dataset** – *pd.DataFrame* object that must contain a *Text* column.
- **dataset_name** – *str* name to use as a save location in the *config.FEATURES_DIRECTORY*.

load()
Method called in flow of *python -m enso.featurize* to prevent loading pre-trained models into memory on file import.

If loading a pre-trained model into memory is not required, *Featurizer.load()* defaults to *pass*.

name()
Prints the name of the current class to aid logging and result formatting.

class enso.featurize.indico_features.**IndicoFinance**(**args*, ***kwargs*)
Featurizer that uses indico's finance features.

featurize(*text*)

Parameters **text** – text of a singular example

Returns *np.ndarray* representation of text

featurize_batch(*X*, *batch_size=32*, ***kwargs*)

Parameters

- **X** – *pd.Series* that contains raw text to featurize
- **batch_size** – int number of examples to process per batch

Returns list of np.ndarray representations of text

generate(*dataset*, *dataset_name*)

Given a *dataset* pandas DataFrame and string *dataset_name*, add column *"Features"* to the provided *pd.DataFrame* and serialize the result to the results folder listed in *config.py*.

If a given featurizer exposes a *featurize_batch()* method, that method will be called to perform featurization. Otherwise, Featurizer's will fall back to calling *featurize()* on each individual example.

Parameters

- **dataset** – *pd.DataFrame* object that must contain a *Text* column.
- **dataset_name** – *str* name to use as a save location in the *config.FEATURES_DIRECTORY*.

load()

Method called in flow of *python -m enso.featurize* to prevent loading pre-trained models into memory on file import.

If loading a pre-trained model into memory is not required, *Featurizer.load()* defaults to *pass*.

name()

Prints the name of the current class to aid logging and result formatting.

class enso.featurize.indico_features.**IndicoTopics**(**args*, ***kwargs*)

Featurizer that uses indico's topics features.

featurize(*text*)

Parameters **text** – text of a singular example

Returns *np.ndarray* representation of text

featurize_batch(*X*, *batch_size=32*, ***kwargs*)

Parameters

- **X** – *pd.Series* that contains raw text to featurize
- **batch_size** – int number of examples to process per batch

Returns list of np.ndarray representations of text

generate(*dataset*, *dataset_name*)

Given a *dataset* pandas DataFrame and string *dataset_name*, add column *"Features"* to the provided *pd.DataFrame* and serialize the result to the results folder listed in *config.py*.

If a given featurizer exposes a *featurize_batch()* method, that method will be called to perform featurization. Otherwise, Featurizer's will fall back to calling *featurize()* on each individual example.

Parameters

- **dataset** – *pd.DataFrame* object that must contain a *Text* column.
- **dataset_name** – *str* name to use as a save location in the *config.FEATURES_DIRECTORY*.

load()

Method called in flow of *python -m enso.featurize* to prevent loading pre-trained models into memory on file import.

If loading a pre-trained model into memory is not required, *Featurizer.load()* defaults to *pass*.

name()

Prints the name of the current class to aid logging and result formatting.

class enso.featurize.indico_features.**IndicoTransformer**(**args*, ***kwargs*)
Featurizer that uses indico's transformer features.

featurize(*text*)

Parameters **text** – text of a singular example

Returns *np.ndarray* representation of text

featurize_batch(*X*, *batch_size=32*, ***kwargs*)

Parameters

- **X** – *pd.Series* that contains raw text to featurize
- **batch_size** – int number of examples to process per batch

Returns list of np.ndarray representations of text

generate(*dataset*, *dataset_name*)
Given a *dataset* pandas DataFrame and string *dataset_name*, add column *"Features"* to the provided *pd.DataFrame* and serialize the result to the results folder listed in *config.py*.

If a given featurizer exposes a *featurize_batch()* method, that method will be called to perform featurization. Otherwise, Featurizer's will fall back to calling *featurize()* on each individual example.

Parameters

- **dataset** – *pd.DataFrame* object that must contain a *Text* column.
- **dataset_name** – *str* name to use as a save location in the *config.FEATURES_DIRECTORY*.

load()
Method called in flow of *python -m enso.featurize* to prevent loading pre-trained models into memory on file import.

If loading a pre-trained model into memory is not required, *Featurizer.load()* defaults to *pass*.

name()
Prints the name of the current class to aid logging and result formatting.

class enso.featurize.indico_features.**IndicoEmotion**(**args*, ***kwargs*)
Featurizer that uses indico's emotion features.

featurize(*text*)

Parameters **text** – text of a singular example

Returns *np.ndarray* representation of text

featurize_batch(*X*, *batch_size=32*, ***kwargs*)

Parameters

- **X** – *pd.Series* that contains raw text to featurize
- **batch_size** – int number of examples to process per batch

Returns list of np.ndarray representations of text

generate(*dataset*, *dataset_name*)
Given a *dataset* pandas DataFrame and string *dataset_name*, add column *"Features"* to the provided *pd.DataFrame* and serialize the result to the results folder listed in *config.py*.

If a given featurizer exposes a *featurize_batch()* method, that method will be called to perform featurization. Otherwise, Featurizer's will fall back to calling *featurize()* on each individual example.

Parameters

- **dataset** – *pd.DataFrame* object that must contain a *Text* column.
- **dataset_name** – *str* name to use as a save location in the *config.FEATURES_DIRECTORY*.

load()
Method called in flow of *python -m enso.featurize* to prevent loading pre-trained models into memory on file import.

If loading a pre-trained model into memory is not required, *Featurizer.load()* defaults to *pass*.

name()
Prints the name of the current class to aid logging and result formatting.

class enso.featurize.indico_features.**IndicoFastText**(**args*, ***kwargs*)
Featurizer that uses indico's fasttext features.

featurize(*text*)

Parameters **text** – text of a singular example

Returns *np.ndarray* representation of text

featurize_batch(*X*, *batch_size=32*, ***kwargs*)

Parameters

- **X** – *pd.Series* that contains raw text to featurize
- **batch_size** – int number of examples to process per batch

Returns list of np.ndarray representations of text

generate(*dataset*, *dataset_name*)
Given a *dataset* pandas DataFrame and string *dataset_name*, add column *"Features"* to the provided *pd.DataFrame* and serialize the result to the results folder listed in *config.py*.

If a given featurizer exposes a *featurize_batch()* method, that method will be called to perform featurization. Otherwise, Featurizer's will fall back to calling *featurize()* on each individual example.

Parameters

- **dataset** – *pd.DataFrame* object that must contain a *Text* column.
- **dataset_name** – *str* name to use as a save location in the *config.FEATURES_DIRECTORY*.

load()
Method called in flow of *python -m enso.featurize* to prevent loading pre-trained models into memory on file import.

If loading a pre-trained model into memory is not required, *Featurizer.load()* defaults to *pass*.

name()
Prints the name of the current class to aid logging and result formatting.

CHAPTER 6

Sampling

Apply a strategy to select training examples to provide to the target model.

6.1 Base Classes

class enso.sample.**Sampler**(*data*, *train_labels*, *train_indices*, *train_size*)
Base class for all *Sampler*'s

Parameters

- **data** – pd.Series of feature vectors
- **train_labels** – pd.Series of targets
- **train_indices** – pd.Series of example indices
- **train_size** – int number of examples to select

sample()
Given the settings provided at initialization, apply the provided sampling strategy

Returns np.array of example indices selected by *Sampler*.

6.2 Included Samplers

class enso.sample.random_sampler.**Random**(*data*, *train_labels*, *train_indices*, *train_size*)
Randomly selects examples from the training dataset.

Parameters

- **data** – pd.Series of feature vectors
- **train_labels** – pd.Series of targets
- **train_indices** – pd.Series of example indices

- **train_size** – int number of examples to select

sample()
Randomly samples feature vectors.

Returns np.array of example indices selected by random sampling

class enso.sample.orthogonal_sampler.**Orthogonal**(*data*, *train_labels*, *train_indices*, *train_size*)
Randomly selects starting points, then selects additional points for which the product of the cosine distance to all starting points is maximally large. Each selected point is then iteratively added to the list of starting points.

Parameters

- **data** – pd.Series of feature vectors
- **train_labels** – pd.Series of targets
- **train_indices** – pd.Series of example indices
- **train_size** – int number of examples to select

sample()
Applies the orthogonal sampling strategy.

Returns np.array of example indices selected by *Orthogonal*.

class enso.sample.kcenter_sampler.**KCenter**(*data*, *train_labels*, *train_indices*, *train_size*)
Randomly selects an example from each class to use as "centers", then selects points that are maximally distant from any given center to use as training examples.

Parameters

- **data** – pd.Series of feature vectors
- **train_labels** – pd.Series of targets
- **train_indices** – pd.Series of example indices
- **train_size** – int number of examples to select

sample()
Applies the KCenter sampling strategy.

Returns np.array of example indices selected by *KCenter*.

CHAPTER 7

Resampling

After a subset of examples is selected, certain examples may be duplicated or removed to adjust the class frequency statistics of the training data.

7.1 Included Resampling Options

enso.resample.**oversample**(*X*, *y*, *max_ratio=50*)

Ensure each class occurs with approximately even frequency in the training set by duplicating examples from relatively rare classes.

Parameters

- **X** – *np.ndarray* of input features
- **y** – *np.ndarray* of corresponding targets
- **max_ratio** – input examples should be duplicated no more than this amount

CHAPTER 8

Target Model Training

After featurization and sampling, a target model is trained on the selected training data. Each *Experiment* must be self-contained – if hyperparameter selection is required, it should be packaged as part of the *Experiment* child class.

8.1 Base Classes

class enso.experiment.**Experiment**(**args*, ***kwargs*)

Base class for all *Experiment*'s.

If hyperparameter selection is necessary for a given target model, the *Experiment* is responsible for performing hyperparameter selection withing the context of *fit()*.

fit(*X*, *y*)

Method to begin training of a given target model provided a set of input features and corresponding targets.

Parameters

- **X** – *np.ndarray* of input features sampled from training data.
- **y** – *np.ndarray* of corresponding targets sampled from training data.

name()

Prints the name of the current class to aid logging and result formatting.

predict(*X*)

Produce a *pd.DataFrame* object contains target model predictions.

Parameters **X** – *np.ndarray* of input features from test data.

Returns *pd.DataFrame* of target model predictions

Prediction format is dependant on class of experiment.

class enso.experiment.**ClassificationExperiment**(**args*, ***kwargs*)

Base class for classification experiments.

fit(*X*, *y*)
Method to begin training of a given target model provided a set of input features and corresponding targets.

Parameters

- **X** – *np.ndarray* of input features sampled from training data.
- **y** – *np.ndarray* of corresponding targets sampled from training data.

name()
Prints the name of the current class to aid logging and result formatting.

predict(*X*)
Produce a *pd.DataFrame* object that maps class labels to class probabilities given test inputs.

Parameters **X** – *np.ndarray* of input features from test data.

Returns *pd.DataFrame* object. Each column should represent a class, and each row should represent an array of probabilities across classes.

class enso.experiment.grid_search.**GridSearch**(**args*, ***kwargs*)
Base class for classification models that select hyperparameters via cross validation. Assumes the *base_model* property set on child classes inherits from *sklearn.base.BaseEstimator* and implements *predict_proba* and *score*.

Variables

- **base_model** – Class name of base model, must be set by child classes.
- **param_grid** – Dictionary that maps from class paramaters to an array of values to search over. Must be set by child classes.

fit(*X*, *y*)
Runs grid search over *self.param_grid* on *self.base_model* to optimize hyper-parameters using KFolds cross-validation, then retrains using the selected parameters on the full training set.

Parameters

- **X** – *np.ndarray* of input features sampled from training data.
- **y** – *np.ndarray* of corresponding targets sampled from training data.

name()
Prints the name of the current class to aid logging and result formatting.

predict(*X*, ***kwargs*)
Predict results on test set based on current internal model.

8.2 Included Experiments:

class enso.experiment.logistic_regression.**LogisticRegressionCV**(**args*, ***kwargs*)
Implementation of a grid-search optimized Logistic Regression model.

fit(*X*, *y*)
Runs grid search over *self.param_grid* on *self.base_model* to optimize hyper-parameters using KFolds cross-validation, then retrains using the selected parameters on the full training set.

Parameters

- **X** – *np.ndarray* of input features sampled from training data.
- **y** – *np.ndarray* of corresponding targets sampled from training data.

name()
Prints the name of the current class to aid logging and result formatting.

predict(*X*, ***kwargs*)
Predict results on test set based on current internal model.

class enso.experiment.naive_bayes.**NaiveBayes**(**args*, ***kwargs*)
Gaussian naive bayes model.

name()
Prints the name of the current class to aid logging and result formatting.

predict(*X*, ***kwargs*)
Predict results on test set based on current internal model.

class enso.experiment.random_forest.**RandomForestCV**(**args*, ***kwargs*)
Implementation of a grid-search optimized RandomForest.

fit(*X*, *y*)
Runs grid search over *self.param_grid* on *self.base_model* to optimize hyper-parameters using KFolds cross-validation, then retrains using the selected parameters on the full training set.

Parameters

- **X** – *np.ndarray* of input features sampled from training data.
- **y** – *np.ndarray* of corresponding targets sampled from training data.

name()
Prints the name of the current class to aid logging and result formatting.

predict(*X*, ***kwargs*)
Predict results on test set based on current internal model.

class enso.experiment.svm.**SupportVectorMachineCV**(**args*, ***kwargs*)
Implementation of a grid-search optimized RBF-SVM.

fit(*X*, *y*)
Runs grid search over *self.param_grid* on *self.base_model* to optimize hyper-parameters using KFolds cross-validation, then retrains using the selected parameters on the full training set.

Parameters

- **X** – *np.ndarray* of input features sampled from training data.
- **y** – *np.ndarray* of corresponding targets sampled from training data.

name()
Prints the name of the current class to aid logging and result formatting.

predict(*X*, ***kwargs*)
Predict results on test set based on current internal model.

CHAPTER 9

Visualization of Results

9.1 Base Classes

class enso.visualize.**Visualizer**(**args*, ***kwargs*)

Base class for creating visualizations.

name()

Prints the name of the current class to aid logging and result formatting.

visualize(*results*, ***kwargs*)

Create visualization for the given test_run. Setting *display* to *True* will default to showing the generated visualizations as they are created, while setting *write* to *True* will default to saving the generated image in the Results directory.

Parameters **results** – pd.DataFrame of results, loaded from results .csv file.

9.2 Included Visualizers

class enso.visualize.facets.**FacetGridVisualizer**(**args*, ***kwargs*)

Create a grid of line graphs based on the value of *config.VISUALIZATION_OPTIONS*

handle_categories(*func*)

Execute a category strategy on a result set. Force the user to make a choice about handling predictions for different classes. It should support either being one of the axes for the Main Visualization, or there should be a strategy for turning multiple entries into a single one

handle_cv(*func*)

Execute a cv strategy on a result_set. Same as handle_categories, but for multiple cv runs rather than multiple classes

name()

Prints the name of the current class to aid logging and result formatting.

visualize(*results*, *x_tile*, *y_tile*, *x_axis*, *y_axis*, *lines*, *results_id=None*, *filename='FacetGridVisualizer'*, ***kwargs*)

Create a tiled visualization of experiment results.

Parameters

- **results** – pd.DataFrame of results, loaded from results .csv file.
- **x_tile** – string name of DataFrame column to vary over the x axis of the grid of line graphs
- **y_tile** – string name of DataFrame column to vary over the y axis of the grid of line graphs
- **x_axis** – string name of DataFrame column to plot on the x axis within each individual line graph
- **y_axis** – string name of DataFrame column to plot on the y axis within each individual line graph
- **lines** – string name or list of DataFrame column string names displayed as separate lines within each graph. Providing multiple values means that each unique combination of values will be displayed as a single line.
- **results_id** – string name of folder to save resulting visual in, relative to the root of the results directory
- **filename** – filename (excluding filetype) to use when saving visualization. Value is relative to folder specified by results_id.

Python Module Index

e

Index

K

L

N

O

P

R

S

V