lime Documentation

Release 0.1

Marco Tulio Ribeiro

Jun 03, 2021

Contents

In this page, you can find the Python API reference for the lime package (local interpretable model-agnostic explanations). For tutorials and more information, visit the github page.

CHAPTER 1

lime package

1.1 Subpackages

1.2 Submodules

1.3 lime.discretize module

Discretizers classes, to be used in lime_tabular

class lime.discretize.**BaseDiscretizer**(*data*, *categorical_features*, *feature_names*, *labels=None*, *random_state=None*, *data_stats=None*)

Bases: object

Abstract class - Build a class that inherits from this class to implement a custom discretizer. Method bins() is to be redefined in the child class, as it is the actual custom part of the discretizer.

Initializer :param data: numpy 2d array :param categorical_features: list of indices (ints) corresponding to the

categorical columns. These features will not be discretized. Everything else will be considered continuous, and will be discretized.

Parameters

- **categorical_names** – map from int to list of names, where categorical_names[x][y] represents the name of the yth value of column x.
- **feature_names** – list of names (strings) corresponding to the columns in the training data.
- **data_stats** – must have 'means', 'stds', 'mins' and 'maxs', use this if you don't want these values to be computed from data

bins(*data*, *labels*)

To be overridden Returns for each feature to discretize the boundaries that form each bin of the discretizer

discretize(*data*)
Discretizes the data. :param data: numpy 2d or 1d array

Returns numpy array of same dimension, discretized.

get_undiscretize_values(*feature*, *values*)

undiscretize(*data*)

class lime.discretize.**DecileDiscretizer**(*data*, *categorical_features*, *feature_names*, *labels=None*, *random_state=None*)
Bases: *lime.discretize.BaseDiscretizer*

bins(*data*, *labels*)
To be overridden Returns for each feature to discretize the boundaries that form each bin of the discretizer

class lime.discretize.**EntropyDiscretizer**(*data*, *categorical_features*, *feature_names*, *labels=None*, *random_state=None*)
Bases: *lime.discretize.BaseDiscretizer*

bins(*data*, *labels*)
To be overridden Returns for each feature to discretize the boundaries that form each bin of the discretizer

class lime.discretize.**QuartileDiscretizer**(*data*, *categorical_features*, *feature_names*, *labels=None*, *random_state=None*)
Bases: *lime.discretize.BaseDiscretizer*

bins(*data*, *labels*)
To be overridden Returns for each feature to discretize the boundaries that form each bin of the discretizer

class lime.discretize.**StatsDiscretizer**(*data*, *categorical_features*, *feature_names*, *labels=None*, *random_state=None*, *data_stats=None*)
Bases: *lime.discretize.BaseDiscretizer*

Class to be used to supply the data stats info when discretize_continuous is true

bins(*data*, *labels*)
To be overridden Returns for each feature to discretize the boundaries that form each bin of the discretizer

1.4 lime.exceptions module

exception lime.exceptions.**LimeError**
Bases: Exception

Raise for errors

1.5 lime.explanation module

Explanation class, with visualization functions.

class lime.explanation.**DomainMapper**
Bases: object

Class for mapping features to the specific domain.

The idea is that there would be a subclass for each domain (text, tables, images, etc), so that we can have a general Explanation class, and separate out the specifics of visualizing features in here.

map_exp_ids(*exp*, ***kwargs*)
Maps the feature ids to concrete names.

Default behaviour is the identity function. Subclasses can implement this as they see fit.

Parameters

- **exp** – list of tuples [(id, weight), (id,weight)]
- **kwargs** – optional keyword arguments

Returns list of tuples [(name, weight), (name, weight)...]

Return type exp

visualize_instance_html(*exp*, *label*, *div_name*, *exp_object_name*, ***kwargs*)
Produces html for visualizing the instance.

Default behaviour does nothing. Subclasses can implement this as they see fit.

Parameters

- **exp** – list of tuples [(id, weight), (id,weight)]
- **label** – label id (integer)
- **div_name** – name of div object to be used for rendering(in js)
- **exp_object_name** – name of js explanation object
- **kwargs** – optional keyword arguments

Returns js code for visualizing the instance

class lime.explanation.**Explanation**(*domain_mapper*, *mode='classification'*, *class_names=None*, *random_state=None*)
Bases: object

Object returned by explainers.

Initializer.

Parameters

- **domain_mapper** – must inherit from DomainMapper class
- **type** – "classification" or "regression"
- **class_names** – list of class names (only used for classification)
- **random_state** – an integer or numpy.RandomState that will be used to generate random numbers. If None, the random state will be initialized using the internal numpy seed.

as_html(*labels=None*, *predict_proba=True*, *show_predicted_value=True*, ***kwargs*)
Returns the explanation as an html page.

Parameters

- **labels** – desired labels to show explanations for (as barcharts). If you ask for a label for which an explanation wasn't computed, will throw an exception. If None, will show explanations for all available labels. (only used for classification)
- **predict_proba** – if true, add barchart with prediction probabilities for the top classes. (only used for classification)
- **show_predicted_value** – if true, add barchart with expected value (only used for regression)
- **kwargs** – keyword arguments, passed to domain_mapper

Returns code for an html page, including javascript includes.

as_list(*label=1*, ***kwargs*)
Returns the explanation as a list.

Parameters

- **label** – desired label. If you ask for a label for which an explanation wasn't computed, will throw an exception. Will be ignored for regression explanations.
- **kwargs** – keyword arguments, passed to domain_mapper

Returns list of tuples (representation, weight), where representation is given by domain_mapper. Weight is a float.

as_map()
Returns the map of explanations.

Returns Map from label to list of tuples (feature_id, weight).

as_pyplot_figure(*label=1*, ***kwargs*)
Returns the explanation as a pyplot figure.

Will throw an error if you don't have matplotlib installed :param label: desired label. If you ask for a label for which an

explanation wasn't computed, will throw an exception. Will be ignored for regression explanations.

Parameters **kwargs** – keyword arguments, passed to domain_mapper

Returns pyplot figure (barchart).

available_labels()
Returns the list of classification labels for which we have any explanations.

save_to_file(*file_path*, *labels=None*, *predict_proba=True*, *show_predicted_value=True*, ***kwargs*)
Saves html explanation to file. .

Params: file_path: file to save explanations to

See as_html() for additional parameters.

show_in_notebook(*labels=None*, *predict_proba=True*, *show_predicted_value=True*, ***kwargs*)
Shows html explanation in ipython notebook.

See as_html() for parameters. This will throw an error if you don't have IPython installed

lime.explanation.**id_generator**(*size=15*, *random_state=None*)
Helper function to generate random div ids. This is useful for embedding HTML into ipython notebooks.

1.6 lime.lime_base module

Contains abstract functionality for learning locally linear sparse model.

class lime.lime_base.**LimeBase**(*kernel_fn*, *verbose=False*, *random_state=None*)
Bases: object

Class for learning a locally linear sparse model from perturbed data

Init function

Parameters

- **kernel_fn** – function that transforms an array of distances into an array of proximity values (floats).
- **verbose** – if true, print local prediction values from linear model.
- **random_state** – an integer or numpy.RandomState that will be used to generate random numbers. If None, the random state will be initialized using the internal numpy seed.

explain_instance_with_data(*neighborhood_data*, *neighborhood_labels*, *distances*, *label*, *num_features*, *feature_selection='auto'*, *model_regressor=None*)

Takes perturbed data, labels and distances, returns explanation.

Parameters

- **neighborhood_data** – perturbed data, 2d array. first element is assumed to be the original data point.
- **neighborhood_labels** – corresponding perturbed labels. should have as many columns as the number of possible labels.
- **distances** – distances to original data point.
- **label** – label for which we want an explanation
- **num_features** – maximum number of features in explanation
- **feature_selection** – how to select num_features. options are: 'forward_selection': iteratively add features to the model.

  This is costly when num_features is high

  'highest_weights': selects the features that have the highest product of absolute weight * original data point when learning with all the features

  'lasso_path': chooses features based on the lasso regularization path

  'none': uses all features, ignores num_features 'auto': uses forward_selection if num_features <= 6, and

  'highest_weights' otherwise.
- **model_regressor** – sklearn regressor to use in explanation. Defaults to Ridge regression if None. Must have **model_regressor.coef_** and 'sample_weight' as a parameter to model_regressor.fit()

Returns intercept is a float. exp is a sorted list of tuples, where each tuple (x,y) corresponds to the feature id (x) and the local weight (y). The list is sorted by decreasing absolute value of y. score is the R^2 value of the returned explanation local_pred is the prediction of the explanation model on the original instance

Return type (intercept, exp, score, local_pred)

feature_selection(*data*, *labels*, *weights*, *num_features*, *method*)

Selects features for the model. see explain_instance_with_data to understand the parameters.

forward_selection(*data*, *labels*, *weights*, *num_features*)

Iteratively adds features to the model

static generate_lars_path(*weighted_data*, *weighted_labels*)

Generates the lars path for weighted data.

Parameters

- **weighted_data** – data that has been weighted by kernel
- **weighted_label** – labels, weighted by kernel

Returns (alphas, coefs), both are arrays corresponding to the regularization parameter and coefficients, respectively

1.7 lime.lime_image module

Functions for explaining classifiers that use Image data.

class lime.lime_image.**ImageExplanation**(*image*, *segments*)

Bases: object

Init function.

Parameters

- **image** – 3d numpy array
- **segments** – 2d numpy array, with the output from skimage.segmentation

get_image_and_mask(*label*, *positive_only=True*, *negative_only=False*, *hide_rest=False*, *num_features=5*, *min_weight=0.0*)

Init function.

Parameters

- **label** – label to explain
- **positive_only** – if True, only take superpixels that positively contribute to the prediction of the label.
- **negative_only** – if True, only take superpixels that negatively contribute to the prediction of the label. If false, and so is positive_only, then both negativey and positively contributions will be taken. Both can't be True at the same time
- **hide_rest** – if True, make the non-explanation part of the return image gray
- **num_features** – number of superpixels to include in explanation
- **min_weight** – minimum weight of the superpixels to include in explanation

Returns (image, mask), where image is a 3d numpy array and mask is a 2d numpy array that can be used with skimage.segmentation.mark_boundaries

class lime.lime_image.**LimeImageExplainer**(*kernel_width=0.25*, *kernel=None*, *verbose=False*, *feature_selection='auto'*, *random_state=None*)

Bases: object

Explains predictions on Image (i.e. matrix) data. For numerical features, perturb them by sampling from a Normal(0,1) and doing the inverse operation of mean-centering and scaling, according to the means and stds in the training data. For categorical features, perturb by sampling according to the training distribution, and making a binary feature that is 1 when the value is the same as the instance being explained.

Init function.

Parameters

- **kernel_width** – kernel width for the exponential kernel.
- **None, defaults to sqrt**(*If*) –

- **kernel** – similarity kernel that takes euclidean distances and kernel width as input and outputs weights in (0,1). If None, defaults to an exponential kernel.
- **verbose** – if true, print local prediction values from linear model
- **feature_selection** – feature selection method. can be 'forward_selection', 'lasso_path', 'none' or 'auto'. See function 'explain_instance_with_data' in lime_base.py for details on what each of the options does.
- **random_state** – an integer or numpy.RandomState that will be used to generate random numbers. If None, the random state will be initialized using the internal numpy seed.

data_labels(*image*, *fudged_image*, *segments*, *classifier_fn*, *num_samples*, *batch_size=10*, *progress_bar=True*)

Generates images and predictions in the neighborhood of this image.

Parameters

- **image** – 3d numpy array, the image
- **fudged_image** – 3d numpy array, image to replace original image when superpixel is turned off
- **segments** – segmentation of the image
- **classifier_fn** – function that takes a list of images and returns a matrix of prediction probabilities
- **num_samples** – size of the neighborhood to learn the linear model
- **batch_size** – classifier_fn will be called on batches of this size.
- **progress_bar** – if True, show tqdm progress bar.

Returns data: dense num_samples * num_superpixels labels: prediction probabilities matrix

Return type A tuple (data, labels), where

explain_instance(*image*, *classifier_fn*, *labels=(1,)*, *hide_color=None*, *top_labels=5*, *num_features=100000*, *num_samples=1000*, *batch_size=10*, *segmentation_fn=None*, *distance_metric='cosine'*, *model_regressor=None*, *random_seed=None*, *progress_bar=True*)

Generates explanations for a prediction.

First, we generate neighborhood data by randomly perturbing features from the instance (see __data_inverse). We then learn locally weighted linear models on this neighborhood data to explain each of the classes in an interpretable way (see lime_base.py).

Parameters

- **image** – 3 dimension RGB image. If this is only two dimensional, we will assume it's a grayscale image and call gray2rgb.
- **classifier_fn** – classifier prediction probability function, which takes a numpy array and outputs prediction probabilities. For ScikitClassifiers , this is classifier.predict_proba.
- **labels** – iterable with labels to be explained.
- **hide_color** – TODO
- **top_labels** – if not None, ignore labels and produce explanations for the K labels with highest prediction probabilities, where K is this parameter.
- **num_features** – maximum number of features present in explanation
- **num_samples** – size of the neighborhood to learn the linear model

- **batch_size** – TODO
- **distance_metric** – the distance metric to use for weights.
- **model_regressor** – sklearn regressor to use in explanation. Defaults
- **Ridge regression in LimeBase. Must have model_regressor.coef** (*to*) –
- **'sample_weight' as a parameter to model_regressor.fit()** (*and*) –
- **segmentation_fn** – SegmentationAlgorithm, wrapped skimage
- **function** (*segmentation*) –
- **random_seed** – integer used as random seed for the segmentation algorithm. If None, a random integer, between 0 and 1000, will be generated using the internal random number generator.
- **progress_bar** – if True, show tqdm progress bar.

Returns An ImageExplanation object (see lime_image.py) with the corresponding explanations.

1.8 lime.lime_tabular module

Functions for explaining classifiers that use tabular data (matrices).

class lime.lime_tabular.**LimeTabularExplainer**(*training_data*, *mode='classification'*, *training_labels=None*, *feature_names=None*, *categorical_features=None*, *categorical_names=None*, *kernel_width=None*, *kernel=None*, *verbose=False*, *class_names=None*, *feature_selection='auto'*, *discretize_continuous=True*, *discretizer='quartile'*, *sample_around_instance=False*, *random_state=None*, *training_data_stats=None*)

Bases: object

Explains predictions on tabular (i.e. matrix) data. For numerical features, perturb them by sampling from a Normal(0,1) and doing the inverse operation of mean-centering and scaling, according to the means and stds in the training data. For categorical features, perturb by sampling according to the training distribution, and making a binary feature that is 1 when the value is the same as the instance being explained.

Init function.

Parameters

- **training_data** – numpy 2d array
- **mode** – "classification" or "regression"
- **training_labels** – labels for training data. Not required, but may be used by discretizer.
- **feature_names** – list of names (strings) corresponding to the columns in the training data.

- **categorical_features** – list of indices (ints) corresponding to the categorical columns. Everything else will be considered continuous. Values in these columns MUST be integers.
- **categorical_names** – map from int to list of names, where categorical_names[x][y] represents the name of the yth value of column x.
- **kernel_width** – kernel width for the exponential kernel. If None, defaults to sqrt (number of columns) * 0.75
- **kernel** – similarity kernel that takes euclidean distances and kernel width as input and outputs weights in (0,1). If None, defaults to an exponential kernel.
- **verbose** – if true, print local prediction values from linear model
- **class_names** – list of class names, ordered according to whatever the classifier is using. If not present, class names will be '0', '1', ...
- **feature_selection** – feature selection method. can be 'forward_selection', 'lasso_path', 'none' or 'auto'. See function 'explain_instance_with_data' in lime_base.py for details on what each of the options does.
- **discretize_continuous** – if True, all non-categorical features will be discretized into quartiles.
- **discretizer** – only matters if discretize_continuous is True and data is not sparse. Options are 'quartile', 'decile', 'entropy' or a BaseDiscretizer instance.
- **sample_around_instance** – if True, will sample continuous features in perturbed samples from a normal centered at the instance being explained. Otherwise, the normal is centered on the mean of the feature data.
- **random_state** – an integer or numpy.RandomState that will be used to generate random numbers. If None, the random state will be initialized using the internal numpy seed.
- **training_data_stats** – a dict object having the details of training data statistics. If None, training data information will be used, only matters if discretize_continuous is True. Must have the following keys: means", "mins", "maxs", "stds", "feature_values", "feature_frequencies"

static convert_and_round(*values*)

explain_instance(*data_row*, *predict_fn*, *labels=(1,)*, *top_labels=None*, *num_features=10*, *num_samples=5000*, *distance_metric='euclidean'*, *model_regressor=None*, *sampling_method='gaussian'*)

Generates explanations for a prediction.

First, we generate neighborhood data by randomly perturbing features from the instance (see __data_inverse). We then learn locally weighted linear models on this neighborhood data to explain each of the classes in an interpretable way (see lime_base.py).

Parameters

- **data_row** – 1d numpy array or scipy.sparse matrix, corresponding to a row
- **predict_fn** – prediction function. For classifiers, this should be a function that takes a numpy array and outputs prediction probabilities. For regressors, this takes a numpy array and returns the predictions. For ScikitClassifiers, this is *classifier.predict_proba()*. For ScikitRegressors, this is *regressor.predict()*. The prediction function needs to work on multiple feature vectors (the vectors randomly perturbed from the data_row).
- **labels** – iterable with labels to be explained.

- **top_labels** – if not None, ignore labels and produce explanations for the K labels with highest prediction probabilities, where K is this parameter.
- **num_features** – maximum number of features present in explanation
- **num_samples** – size of the neighborhood to learn the linear model
- **distance_metric** – the distance metric to use for weights.
- **model_regressor** – sklearn regressor to use in explanation. Defaults to Ridge regression in LimeBase. Must have **model_regressor.coef_** and 'sample_weight' as a parameter to model_regressor.fit()
- **sampling_method** – Method to sample synthetic data. Defaults to Gaussian sampling. Can also use Latin Hypercube Sampling.

Returns An Explanation object (see explanation.py) with the corresponding explanations.

static validate_training_data_stats(*training_data_stats*)
Method to validate the structure of training data stats

class lime.lime_tabular.**RecurrentTabularExplainer**(*training_data*, *mode='classification'*, *training_labels=None*, *feature_names=None*, *categorical_features=None*, *categorical_names=None*, *kernel_width=None*, *kernel=None*, *verbose=False*, *class_names=None*, *feature_selection='auto'*, *discretize_continuous=True*, *discretizer='quartile'*, *random_state=None*)

Bases: *lime.lime_tabular.LimeTabularExplainer*

An explainer for keras-style recurrent neural networks, where the input shape is (n_samples, n_timesteps, n_features). This class just extends the LimeTabularExplainer class and reshapes the training data and feature names such that they become something like

(val1_t1, val1_t2, val1_t3, ..., val2_t1, ..., valn_tn)

Each of the methods that take data reshape it appropriately, so you can pass in the training/testing data exactly as you would to the recurrent neural network.

Parameters

- **training_data** – numpy 3d array with shape (n_samples, n_timesteps, n_features)
- **mode** – "classification" or "regression"
- **training_labels** – labels for training data. Not required, but may be used by discretizer.
- **feature_names** – list of names (strings) corresponding to the columns in the training data.
- **categorical_features** – list of indices (ints) corresponding to the categorical columns. Everything else will be considered continuous. Values in these columns MUST be integers.
- **categorical_names** – map from int to list of names, where categorical_names[x][y] represents the name of the yth value of column x.
- **kernel_width** – kernel width for the exponential kernel.

- **None, defaults to sqrt** (*If*) –
- **kernel** – similarity kernel that takes euclidean distances and kernel width as input and outputs weights in (0,1). If None, defaults to an exponential kernel.
- **verbose** – if true, print local prediction values from linear model
- **class_names** – list of class names, ordered according to whatever the classifier is using. If not present, class names will be '0', '1', ...
- **feature_selection** – feature selection method. can be 'forward_selection', 'lasso_path', 'none' or 'auto'. See function 'explain_instance_with_data' in lime_base.py for details on what each of the options does.
- **discretize_continuous** – if True, all non-categorical features will be discretized into quartiles.
- **discretizer** – only matters if discretize_continuous is True. Options are 'quartile', 'decile', 'entropy' or a BaseDiscretizer instance.
- **random_state** – an integer or numpy.RandomState that will be used to generate random numbers. If None, the random state will be initialized using the internal numpy seed.

explain_instance (*data_row*, *classifier_fn*, *labels=(1,)*, *top_labels=None*, *num_features=10*, *num_samples=5000*, *distance_metric='euclidean'*, *model_regressor=None*)

Generates explanations for a prediction.

First, we generate neighborhood data by randomly perturbing features from the instance (see __data_inverse). We then learn locally weighted linear models on this neighborhood data to explain each of the classes in an interpretable way (see lime_base.py).

Parameters

- **data_row** – 2d numpy array, corresponding to a row
- **classifier_fn** – classifier prediction probability function, which takes a numpy array and outputs prediction probabilities. For ScikitClassifiers , this is classifier.predict_proba.
- **labels** – iterable with labels to be explained.
- **top_labels** – if not None, ignore labels and produce explanations for the K labels with highest prediction probabilities, where K is this parameter.
- **num_features** – maximum number of features present in explanation
- **num_samples** – size of the neighborhood to learn the linear model
- **distance_metric** – the distance metric to use for weights.
- **model_regressor** – sklearn regressor to use in explanation. Defaults to Ridge regression in LimeBase. Must have **model_regressor.coef_** and 'sample_weight' as a parameter to model_regressor.fit()

Returns An Explanation object (see explanation.py) with the corresponding explanations.

class lime.lime_tabular.**TableDomainMapper** (*feature_names*, *feature_values*, *scaled_row*, *categorical_features*, *discretized_feature_names=None*, *feature_indexes=None*)

Bases: *lime.explanation.DomainMapper*

Maps feature ids to names, generates table views, etc

Init.

Parameters

- **feature_names** – list of feature names, in order
- **feature_values** – list of strings with the values of the original row
- **scaled_row** – scaled row
- **categorical_features** – list of categorical features ids (ints)
- **feature_indexes** – optional feature indexes used in the sparse case

map_exp_ids(*exp*)
Maps ids to feature names.

Parameters **exp** – list of tuples [(id, weight), (id,weight)]

Returns list of tuples (feature_name, weight)

visualize_instance_html(*exp*, *label*, *div_name*, *exp_object_name*, *show_table=True*, *show_all=False*)
Shows the current example in a table format.

Parameters

- **exp** – list of tuples [(id, weight), (id,weight)]
- **label** – label id (integer)
- **div_name** – name of div object to be used for rendering(in js)
- **exp_object_name** – name of js explanation object
- **show_table** – if False, don't show table visualization.
- **show_all** – if True, show zero-weighted features in the table.

1.9 lime.lime_text module

Functions for explaining text classifiers.

class lime.lime_text.**IndexedCharacters**(*raw_string*, *bow=True*, *mask_string=None*)
Bases: object

String with various indexes.

Initializer.

Parameters

- **raw_string** – string with raw text in it
- **bow** – if True, a char is the same everywhere in the text - i.e. we will index multiple occurrences of the same character. If False, order matters, so that the same word will have different ids according to position.
- **mask_string** – If not None, replace characters with this if bow=False if None, default value is chr(0)

inverse_removing(*words_to_remove*)
Returns a string after removing the appropriate words.

If self.bow is false, replaces word with UNKWORDZ instead of removing it.

Parameters **words_to_remove** – list of ids (ints) to remove

Returns original raw string with appropriate words removed.

num_words()
Returns the number of tokens in the vocabulary for this document.

raw_string()
Returns the original raw string

string_position(*id_*)
Returns a np array with indices to **id_** (int) occurrences

word(*id_*)
Returns the word that corresponds to **id_** (int)

class lime.lime_text.**IndexedString**(*raw_string*, *split_expression='\W+'*, *bow=True*, *mask_string=None*)
Bases: object

String with various indexes.

Initializer.

Parameters

- **raw_string** – string with raw text in it
- **split_expression** – Regex string or callable. If regex string, will be used with re.split. If callable, the function should return a list of tokens.
- **bow** – if True, a word is the same everywhere in the text - i.e. we will index multiple occurrences of the same word. If False, order matters, so that the same word will have different ids according to position.
- **mask_string** – If not None, replace words with this if bow=False if None, default value is UNKWORDZ

inverse_removing(*words_to_remove*)
Returns a string after removing the appropriate words.

If self.bow is false, replaces word with UNKWORDZ instead of removing it.

Parameters **words_to_remove** – list of ids (ints) to remove

Returns original raw string with appropriate words removed.

num_words()
Returns the number of tokens in the vocabulary for this document.

raw_string()
Returns the original raw string

string_position(*id_*)
Returns a np array with indices to **id_** (int) occurrences

word(*id_*)
Returns the word that corresponds to **id_** (int)

class lime.lime_text.**LimeTextExplainer**(*kernel_width=25*, *kernel=None*, *verbose=False*, *class_names=None*, *feature_selection='auto'*, *split_expression='\W+'*, *bow=True*, *mask_string=None*, *random_state=None*, *char_level=False*)
Bases: object

Explains text classifiers. Currently, we are using an exponential kernel on cosine distance, and restricting explanations to words that are present in documents.

Init function.

Parameters

- **kernel_width** – kernel width for the exponential kernel.
- **kernel** – similarity kernel that takes euclidean distances and kernel width as input and outputs weights in (0,1). If None, defaults to an exponential kernel.
- **verbose** – if true, print local prediction values from linear model
- **class_names** – list of class names, ordered according to whatever the classifier is using. If not present, class names will be '0', '1', ...
- **feature_selection** – feature selection method. can be 'forward_selection', 'lasso_path', 'none' or 'auto'. See function 'explain_instance_with_data' in lime_base.py for details on what each of the options does.
- **split_expression** – Regex string or callable. If regex string, will be used with re.split. If callable, the function should return a list of tokens.
- **bow** – if True (bag of words), will perturb input data by removing all occurrences of individual words or characters. Explanations will be in terms of these words. Otherwise, will explain in terms of word-positions, so that a word may be important the first time it appears and unimportant the second. Only set to false if the classifier uses word order in some way (bigrams, etc), or if you set char_level=True.
- **mask_string** – String used to mask tokens or characters if bow=False if None, will be 'UNKWORDZ' if char_level=False, chr(0) otherwise.
- **random_state** – an integer or numpy.RandomState that will be used to generate random numbers. If None, the random state will be initialized using the internal numpy seed.
- **char_level** – an boolean identifying that we treat each character as an independent occurence in the string

explain_instance(*text_instance*, *classifier_fn*, *labels=(1,)*, *top_labels=None*, *num_features=10*, *num_samples=5000*, *distance_metric='cosine'*, *model_regressor=None*)

Generates explanations for a prediction.

First, we generate neighborhood data by randomly hiding features from the instance (see __data_labels_distance_mapping). We then learn locally weighted linear models on this neighborhood data to explain each of the classes in an interpretable way (see lime_base.py).

Parameters

- **text_instance** – raw text string to be explained.
- **classifier_fn** – classifier prediction probability function, which takes a list of d strings and outputs a (d, k) numpy array with prediction probabilities, where k is the number of classes. For ScikitClassifiers , this is classifier.predict_proba.
- **labels** – iterable with labels to be explained.
- **top_labels** – if not None, ignore labels and produce explanations for the K labels with highest prediction probabilities, where K is this parameter.
- **num_features** – maximum number of features present in explanation
- **num_samples** – size of the neighborhood to learn the linear model
- **distance_metric** – the distance metric to use for sample weighting, defaults to cosine similarity
- **model_regressor** – sklearn regressor to use in explanation. Defaults

- **Ridge regression in LimeBase. Must have model_regressor.coef** (*to*) –
- **'sample_weight' as a parameter to model_regressor.fit()** (*and*) –

Returns An Explanation object (see explanation.py) with the corresponding explanations.

class lime.lime_text.**TextDomainMapper**(*indexed_string*)

Bases: *lime.explanation.DomainMapper*

Maps feature ids to words or word-positions

Initializer.

Parameters **indexed_string** – lime_text.IndexedString, original string

map_exp_ids(*exp*, *positions=False*)

Maps ids to words or word-position strings.

Parameters

- **exp** – list of tuples [(id, weight), (id,weight)]
- **positions** – if True, also return word positions

Returns list of tuples (word, weight), or (word_positions, weight) if examples: ('bad', 1) or ('bad_3-6-12', 1)

visualize_instance_html(*exp*, *label*, *div_name*, *exp_object_name*, *text=True*, *opacity=True*)

Adds text with highlighted words to visualization.

Parameters

- **exp** – list of tuples [(id, weight), (id,weight)]
- **label** – label id (integer)
- **div_name** – name of div object to be used for rendering(in js)
- **exp_object_name** – name of js explanation object
- **text** – if False, return empty
- **opacity** – if True, fade colors according to weight

1.10 lime.submodular_pick module

class lime.submodular_pick.**SubmodularPick**(*explainer*, *data*, *predict_fn*, *method='sample'*, *sample_size=1000*, *num_exps_desired=5*, *num_features=10*, ***kwargs*)

Bases: object

Class for submodular pick

Saves a representative sample of explanation objects using SP-LIME, as well as saving all generated explanations

First, a collection of candidate explanations are generated (see explain_instance). From these candidates, num_exps_desired are chosen using submodular pick. (see marcotcr et al paper).

Parameters

- **data** – a numpy array where each row is a single input into predict_fn

- **predict_fn** – prediction function. For classifiers, this should be a function that takes a numpy array and outputs prediction probabilities. For regressors, this takes a numpy array and returns the predictions. For ScikitClassifiers, this is *classifier.predict_proba()*. For ScikitRegressors, this is *regressor.predict()*. The prediction function needs to work on multiple feature vectors (the vectors randomly perturbed from the data_row).
- **method** – The method to use to generate candidate explanations method == 'sample' will sample the data uniformly at random. The sample size is given by sample_size. Otherwise if method == 'full' then explanations will be generated for the entire data. l
- **sample_size** – The number of instances to explain if method == 'sample'
- **num_exps_desired** – The number of explanation objects returned
- **num_features** – maximum number of features present in explanation

Sets value: sp_explanations: A list of explanation objects that has a high coverage explanations: All the candidate explanations saved for potential future use.

1.11 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

l

Index

M

N

Q

R

S

T

U

V

W