Soft-Boiled Documentation

Release 0.9.0

Lab41

Mar 21, 2017

Contents

The usage examples below assume that you have created a zip file containing the top level directory of the repo called soft-boiled.zip. To create zip file run "zip -r soft-boiled.zip __init__.py src" from inside the repository directory.

Example IPython notebooks demonstrating the functionality of these algorithms can also be found in the notebooks directory of this repository.

CHAPTER 1

Spatial Label Propagation [slp.py]:

Usage:

```
sc.addPyFile ('/path/to/zip/soft-boiled.zip') # Can be an hdfs path
from src.algorithms import slp

# Create dataframe from parquet data
tweets = sqlCtx.read.parquet('hdfs:///post_etl_datasets/twitter')
tweets.registerTempTable('my_tweets')

# Get Known Locations
locs_known = slp.get_known_locs(sqlCtx, 'my_tweets', min_locs=3,         dispersion_
↪threshold=50, num_partitions=30)

# Get at mention network, bi-directional at mentions
edge_list = slp.get_edge_list(sqlCtx, 'my_tweets')

# Run Spatial label propagation with 5 iterations
estimated_locs = slp.train_slp(filtered_locs_known, edge_list, 5,     dispersion_
↪threshold=100)


# prepare the input functions to the evaluate function. In this case, we create a␣
↪holdout function
# that filter approximately 10% of the data for testing, and we also have a closure␣
↪that prepopulates
# some of the parameters to the train_slp function
holdout_10pct = lambda (src_id): src_id[-1] != '9'
train_f = lambda locs, edges : slp.train_slp(locs, edges, 4, neighbor_threshold=4,␣
↪dispersion_threshold=150)

# Test results
test_results = slp.evaluate(locs_known, edge_list, holdout_10pct, train_f)```
```

Options:

Related to calculating the median point amongst a collection of points:

dispersion_threshold: This is the maximum median distance in km a point can be from the remaining points and still estimate a location

min_locs: Number of geotagged posts that a user must have to be included in ground truth.

Related to the actual label propagation: num_iters: This controls the number of iterations of label propagation performed

CHAPTER 2

Gaussian Mixture Model [gmm.py]

Usage:

```
sc.addPyFile ('/path/to/zip/soft-boiled.zip') # Can be an hdfs path
from src.algorithms import gmm

# Create dataframe from parquet data
tweets = sqlCtx.read.parquet('hdfs:///post_etl_datasets/twitter')
tweets.registerTempTable('my_tweets')

# Train GMM model
gmm_model = gmm.train_gmm(sqlCtx, 'my_tweets', ['user.location', 'text'], min_
↪occurrences=10, max_num_components=12)

# Test GMM model
test_results = gmm.run_gmm_test(sc, sqlCtx, 'my_tweets', ['user.location', 'text'],␣
↪gmm_model)
print test_results

# Use GMM model to predict tweets
other_tweets = sqlCtx.read.parquet('hdfs:///post_etl_datasets/twitter')
estimated_locs = gmm.predict_user_gmm(sc, other_tweets, ['user.location'], gmm_model,␣
↪radius=100, predict_lower_bound=0.2)

# Save model for future prediction use
gmm.save_model(gmm_model, '/local/path/to/model_file.csv.gz')

# Load a model, produces the same output as train
gmm_model = gmm.load_model('/local/path/to/model_file.csv.gz')
```

Options:

Related to GMM:

fields: A set of fields to use to train/test the GMM model. Currently only user.location and text are supported

min_occurrences: Number of times that a token must appear with a known location in the text to be estimated

max_num_components: Limit on the number of GMM components that can be used

Predict User Options:

radius: Predict the probability that the user is within this distance of most likely point, used with predict_lower_bound

predict_lower_bound: Used with radius to filter user location estimates with probability lower than threshold

CHAPTER 3

Contents:

class src.algorithms.slp.**LocEstimate**(*geo_coord*, *dispersion*, *dispersion_std_dev*)

dispersion
Alias for field number 1

dispersion_std_dev
Alias for field number 2

geo_coord
Alias for field number 0

src.algorithms.slp.**evaluate**(*locs_known*, *edges*, *holdout_func*, *slp_closure*)
This function is used to assess various stats regarding how well SLP is running. Given all locs that are known and all edges that are known, this funciton will first apply the holdout to the locs_known, allowing for a ground truth comparison to be used. Then, it applies the non-holdout set to the training function, which should yield the locations of the holdout for comparison.

For example:

```
holdout = lambda (src_id) : src_id[-1] == '6'
trainer = lambda l, e : slp.train_slp(l, e, 3)
results = evaluate(locs_known, edges, holdout, trainer)
```

Parameters

- **locs_known** (*rdd of LocEstimate objects*) – The complete list of locations
- **edges** (*rdd of (src_id, (dest_id, weight))*) – all available edge information
- **holdout_func** (*function*) – function responsible for filtering a holdout data set. For example:

```
lambda (src_id) : src_id[-1] == '6'
```

can be used to get approximately 10% of the data since the src_id's are evenly distributed numeric values

- **slp_closure** (*function closure*) – a closure over the slp train function. For example:

```
lambda locs, edges :

        slp.train_slp(locs, edges, 4, neighbor_threshold=4,␣
↪dispersion_threshold=150)
```

can be used for training with specific threshold parameters

Returns

stats of the results from the SLP algorithm

median: median difference of predicted versus actual

mean: mean difference of predicted versus actual

coverage: ratio of number of predicted locations to number of original unknown locations

reserved_locs: number of known locations used to train

total_locs: number of known locations input into this function

found_locs: number of predicted locations

holdout_ratio: ratio of the holdout set to the entire set

Return type results (dict)

src.algorithms.slp.**get_edge_list**(*sqlCtx*, *table_name*, *num_partitions=300*)

Given a loaded twitter table, this will return the @mention network in the form (src_id, (dest_id, num_@mentions))

Parameters

- **sqlCtx** (*Spark SQL Context*) – A Spark SQL context
- **table_name** (*string*) – Table name that was registered when loading the data
- **num_paritions** (*int*) – Optimizer for specifying the number of paritions for the resulting RDD to use

Returns edges loaded from the table

Return type edges (rdd (src_id, (dest_id, weight)))

src.algorithms.slp.**get_known_locs**(*sqlCtx*, *table_name*, *include_places=True*, *min_locs=3*, *num_partitions=30*, *dispersion_threshold=50*)

Given a loaded twitter table, this will return all the twitter users with locations. A user's location is determined by the median location of all known tweets. A user must have at least min_locs locations in order for a location to be estimated

Parameters

- **sqlCtx** (*Spark SQL Context*) – A Spark SQL context
- **table_name** (*string*) – Table name that was registered when loading the data
- **min_locs** (*int*) – Minimum number tweets that have a location in order to infer a location for the user

- **num_partitions** (*int*) – Optimizer for specifying the number of partitions for the resulting RDD to use.
- **dispersion_threhold** (*int*) – A distance threhold on the dispersion of the estimated location for a user. We consider those estimated points with dispersion greater than the treshold unable to be predicted given how dispersed the tweet distances are from one another.

Returns Found locations of users. This rdd is often used as the ground truth of locations

Return type locations (rdd of LocEstimate)

src.algorithms.slp.**median**(*distance_func*, *vertices*, *weights=None*)
given a python list of vertices, and a distance function, this will find the vertex that is most central relative to all other vertices. All of the vertices must have geocoords

Parameters

- **distance_func** (*function*) – A function to calculate the distance between two GeoCoord objects
- **vertices** (*list*) – List of GeoCoord objects

Returns The median point

Return type *LocEstimate*

src.algorithms.slp.**train_slp**(*locs_known*, *edge_list*, *num_iters*, *neighbor_threshold=3*, *dispersion_threshold=100*)
Core SLP algorithm

Parameters

- **locs_known** (*rdd of LocEstimate objects*) – Locations that are known for the SLP network
- **edge_list** (*rdd of edges (src_id, (dest_id, weight))*) – edges representing the at mention network
- **num_iters** (*int*) – number of iterations to run the algorithm
- **neighbor_threshold** (*int*) – The minimum number of neighbors required in order for SLP to try and predict a location of a node in the network
- **dispersion_theshold** (*int*) – The maximum median distance amoung a local at mention network in order to predict a node's location.

Returns The locations found and known

Return type locations (rdd of LocEstimate objects)

src.algorithms.gmm.**GMMLocEstimate**
alias of LocEstimate

src.algorithms.gmm.**combine_gmms**(*gmms*)
Takes an array of gaussian mixture models and produces a GMM that is the weighted sum of the models

Parameters **gmms** (*list*) – A list of (mixture.GMM, median_error_on_training) models

Returns A single GMM model that is the weighted sum of the input gmm models

Return type new_gmm (mixture.GMM)

src.algorithms.gmm.**fit_gmm_to_locations**(*geo_coords*, *max_num_components*)
Searches within bounts to fit a GMM with the optimal number of components

Parameters

- **geo_coords** (*list*) – A list of GeoCoord points to fit a GMM distribution to
- **max_num_components** (*int*) – The maximum number of components that the GMM model can have

Returns Tuple containing the best mixture.GMM and the error of that model on the training data

Return type gmm_estimate (tuple)

src.algorithms.gmm.**get_errors**(*model*, *points*)
Computes the median error for a GMM model and a set of training points

Parameters

- **model** (*mixture.GMM*) – A GMM model for a word
- **points** (*list*) – A list of (lat, lon) tuples

Returns The median distance to the training points from the most likely point

Return type *median* (float)

src.algorithms.gmm.**get_location_from_tweet**(*row*)
Extract location from a tweet object. If geo.coordinates not present use center of place.bounding_box.

Parameters **row** (*Row*) – A spark sql row containing a tweet

Retruns: GeoCoord: The location in the tweet

src.algorithms.gmm.**get_most_likely_point**(*tokens*, *model_bcast*, *radius=None*)
Create the combined GMM and find the most likely point. This function is called in a flatMap so return a list with 0 or 1 item

Parameters

- **tokens** (*list*) – list of words in tweet
- **model_bcast** (*pyspark.Broadcast*) – A broadcast version of a dictionary of GMM model for the entire vocabulary
- **radius** (*float*) – Distance from most likely point at which we should estimate containment probability (if not None)

Returns A list with 0 or 1 GMMLocEstimates

Return type loc_estimate (list)

src.algorithms.gmm.**load_model**(*input_fname*)
Load a pre-trained model

Parameters **input_fname** (*str*) – Local file path to read GMM model from

Returns A dictionary of the form {word: (mixture.GMM, error)}

Return type model (dict)

src.algorithms.gmm.**predict_probability_area**(*model*, *upper_bound*, *lower_bound*)
Predict the probability that the true location is within a specified bounding box given a GMM model

Parameters

- **model** (*mixture.GMM*) – GMM model to use
- **upper_bound** (*list*) – [upper lat, right lon] of bounding box
- **lower_bound** (*list*) – [lower_lat, left_lon] of bounding box

Returns Probability from 0 to 1 of true location being in bounding box

Return type total_prob (float)

src.algorithms.gmm.**predict_probability_radius**(*gmm_model*, *radius*, *center_point*)
Attempt to estimate the probability that the true location is within some radius of a given center point. Estimate is based on estimating probability in corners of bounding box and subtracting from total probability mass

Parameters

- **gmm_model** (*mixture.GMM*) – GMM model to use
- **radius** (*float*) – Radius from center point to include in estimate
- **center_point** (*tuple*) – (lat, lon) center point

Returns Probability from 0 to 1 of true location being in the specified radius

Return type total_prob (float)

src.algorithms.gmm.**predict_user_gmm**(*sc*, *tweets_to_predict*, *fields*, *model*, *radius=None*, *predict_lower_bound=0*, *num_partitions=5000*)
Takes a set of tweets and for each user in those tweets it predicts a location Also returned are the probability of that prediction location being w/n 100 km of the true point

Parameters

- **sc** (*pyspark.SparkContext*) – Spark Context to use for execution
- **tweets_to_predict** (*RDD*) – RDD of twitter Row objects
- **fields** (*list*) – List of field names to extract and then use for GMM prediction
- **model** (*dict*) – Dictionary of {word:(mixture.GMM, error)}
- **radius** (*float*) – Distance from most likely point at which we should estimate containment probability (if not None)
- **predict_lower_bound** (*float*) – Probability hreshold below which we should filter tweets
- **num_partitions** (*int*) – Number of partitions. Should be based on size of the data

Returns An RDD of (id_str, GMMLocEstimate)

Return type loc_est_by_user (RDD)

src.algorithms.gmm.**run_gmm_test**(*sc*, *sqlCtx*, *table_name*, *fields*, *model*, *where_clause=''*)
Test a pretrained model on a table of test data

Parameters

- **sc** (*pyspark.SparkContext*) – Spark Context to use for execution
- **sqlCtx** (*pyspark.sql.SQLContext*) – Spark SQL Context to use for sql queries
- **table_name** (*str*) – Table name to query for test data
- **fields** (*list*) – List of field names to extract and then use for GMM prediction
- **model** (*dict*) – Dictionary of {word:(mixture.GMM, error)}
- **where_clause** (*str*) – A where clause that can be applied to the query

Returns A description of the performance of the GMM Algorithm

Return type final_result (dict)

src.algorithms.gmm.**save_model**(*model*, *output_fname*)
Save the current model for future use

Parameters

- **model** (*dict*) – A dictionary of the form {word: (mixture.GMM, error)}
- **output_fname** (*str*) – Local file path to store GMM model

src.algorithms.gmm.**tokenize_tweet**(*inputRow*, *fields*)
A simple tokenizer that takes a tweet as input and, splitting on whitespace, and returns words in the tweet

Parameters

- **inputRow** (*Row*) – A spark sql row containing a tweet
- **fields** (*list*) – A list of field names which directs tokenize on which fields to use as source data

Returns A list of words appearing in the tweet

Return type tokens (list)

src.algorithms.gmm.**train_gmm**(*sqlCtx*, *table_name*, *fields*, *min_occurrences=10*, *max_num_components=12*, *where_clause=''*)
Train a set of GMMs for a given set of training data

Parameters

- **sqlCtx** (*pyspark.sql.SQLContext*) – Spark SQL Context to use for sql queries
- **table_name** (*str*) – Table name to query for test data
- **fields** (*list*) – List of field names to extract and then use for GMM prediction
- **min_occurrences** (*int*) – Number of times a word must appear to be incldued in the model
- **max_num_components** (*int*) – The maximum number of components that the GMM model can have
- **where_clause** (*str*) – A where clause that can be applied to the query

Returns Dictionary of {word:(mixture.GMM, error)}

Return type model (dict)

class src.algorithms.estimator.**EstimatorCurve**(*w_stdev*, *wo_stdev*)
The EstimatorCurve class is used to assess the confidence of a predicted location for SLP.

w_stdev
numpy arr – A two dimensional numpy array representing the estimator curve. The x axis is the standard deviations and y axis is the probability. The curve is a CDF. This curve is generated from known locations where at least two neighbors are at different locations.

wo_stdev
numpy_arr – A two dimensional numpy array representing the estimator curve

static build_curve(*vals*, *desired_samples*)
Static helper method for building the curve from a set of stdev stample

Parameters

- **vals** (*rdd of floats*) – The rdd containing the standard deviation from the distance between the estimated location and the actual locationn

- **desired_samples** (*int*) – For larger RDDs it is more efficient to take a sample for the collect

Returns

two dimensional array representing the curve.

Column 0 is the sorted stdevs and column 1 is the percentage for the CDF.

Return type curve (numpy.ndarray)

confidence_estimation_viewer(*sc*, *eval_rdd*)

Displays a plot of the estimated and actual probability that the true point is within an array of radius values

Parameters

- **curve** (*numpy.darray*) – x axis is stdev, y axis is percent
- **eval_rdd** (*rdd (src_id, (dist, loc_estimate))*) – this is the result of the evaluator function

static load_from_file(*name='estimator'*)

Loads an Estimator curve from csv files

Parameters **name** (*string*) – prefix name for the two CSV files

static load_from_rdds(*locs_known*, *edges*, *desired_samples=1000*, *dispersion_threshold=150*, *neighbor_threshold=3*)

Creates an EstimatorCurve

Parameters

- **locs_known** (*rdd of LocEstimate*) – RDD of locations that are known
- **edges** (*rdd of (src_id (dest_id, weight)*) – RDD of edges in the network
- **desired_samples** (*int*) – Limit the curve to just a sample of data

Returns A new EstimatorCurve representing the known input data

Return type *EstimatorCurve*

lookup(*val*, *axis=0*)

static lookup_static(*table*, *val*, *axis=0*)

lookups up closes stdev by subtracting from lookup table, taking absolute value and finding which is closest to zero by sorting and taking the first element

Args: num_std_devs (float): the stdev to lookup

Returns: CDF (float) : Percentage of actual locations found to be within the input stdev

plot(*w_stdev_lim=10*, *wo_stdev_lim=1000*)

Plots both the stdev curve and the distance curve for when the stdev is 0

Parameters

- **w_stdev_lim** (*int*) – x axis limit for the plot
- **wo_stdev_lim** (*int*) – x axis limit for the plot

predict_probability_area(*upper_bound*, *lower_bound*, *estimated_loc*)

Given a prediction and a bounding box this will return a confidence range for that prediction

Parameters

- **upper_bound** (*geoCoord*) – bounding box top right geoCoord
- **lower_bound** (*geoCoord*) – bounding box bottom left geoCoord

- **estimated_loc** (LocEstimate) – geoCoord of the estimated location

Returns A probability range tuple (min probability, max probability)

Return type Probability Tuple(Tuple(float,float))

save(*name='estimator'*)
Saves the EstimatorCurve as a csv

Parameters **name** (*string*) – A prefix name for the filename. Two CSVs will be created– one for when the stdev is 0, and one for when it is greater than 0

validator(*sc*, *eval_rdd*)
Validates a curve

Parameters

- **curve** (*numpy.darray*) – x axis is stdev, y axis is percent
- **eval_rdd** (*rdd (src_id, (dist, loc_estimate))*) – this is the result of the evaluator function

class src.algorithms.estimator.**EstimatorCurve**(*w_stdev*, *wo_stdev*)
The EstimatorCurve class is used to assess the confidence of a predicted location for SLP.

w_stdev
numpy arr – A two dimensional numpy array representing the estimator curve. The x axis is the standard deviations and y axis is the probability. The curve is a CDF. This curve is generated from known locations where at least two neighbors are at different locations.

wo_stdev
numpy_arr – A two dimensional numpy array representing the estimator curve

static build_curve(*vals*, *desired_samples*)
Static helper method for building the curve from a set of stdev stample

Parameters

- **vals** (*rdd of floats*) – The rdd containing the standard deviation from the distance between the estimated location and the actual locationn
- **desired_samples** (*int*) – For larger RDDs it is more efficient to take a sample for the collect

Returns

two dimensional array representing the curve.

Column 0 is the sorted stdevs and column 1 is the percentage for the CDF.

Return type curve (numpy.ndarray)

confidence_estimation_viewer(*sc*, *eval_rdd*)
Displays a plot of the estimated and actual probability that the true point is within an array of radius values

Parameters

- **curve** (*numpy.darray*) – x axis is stdev, y axis is percent
- **eval_rdd** (*rdd (src_id, (dist, loc_estimate))*) – this is the result of the evaluator function

static load_from_file(*name='estimator'*)
Loads an Estimator curve from csv files

Parameters **name** (*string*) – prefix name for the two CSV files

static load_from_rdds(*locs_known*, *edges*, *desired_samples=1000*, *dispersion_threshold=150*, *neighbor_threshold=3*)

Creates an EstimatorCurve

Parameters

- **locs_known** (*rdd of LocEstimate*) – RDD of locations that are known
- **edges** (*rdd of (src_id (dest_id, weight))*) – RDD of edges in the network
- **desired_samples** (*int*) – Limit the curve to just a sample of data

Returns A new EstimatorCurve representing the known input data

Return type *EstimatorCurve*

lookup(*val*, *axis=0*)

static lookup_static(*table*, *val*, *axis=0*)

lookups up closes stdev by subtracting from lookup table, taking absolute value and finding which is closest to zero by sorting and taking the first element

Args: num_std_devs (float): the stdev to lookup

Returns: CDF (float) : Percentage of actual locations found to be within the input stdev

plot(*w_stdev_lim=10*, *wo_stdev_lim=1000*)

Plots both the stdev curve and the distance curve for when the stdev is 0

Parameters

- **w_stdev_lim** (*int*) – x axis limit for the plot
- **wo_stdev_lim** (*int*) – x axis limit for the plot

predict_probability_area(*upper_bound*, *lower_bound*, *estimated_loc*)

Given a prediction and a bounding box this will return a confidence range for that prediction

Parameters

- **upper_bound** (*geoCoord*) – bounding box top right geoCoord
- **lower_bound** (*geoCoord*) – bounding box bottom left geoCoord
- **estimated_loc** (LocEstimate) – geoCoord of the estimated location

Returns A probability range tuple (min probability, max probability)

Return type Probability Tuple(Tuple(float,float))

save(*name='estimator'*)

Saves the EstimatorCurve as a csv

Parameters **name** (*string*) – A prefix name for the filename. Two CSVs will be created– one for when the stdev is 0, and one for when it is greater than 0

validator(*sc*, *eval_rdd*)

Validates a curve

Parameters

- **curve** (*numpy.darray*) – x axis is stdev, y axis is percent
- **eval_rdd** (*rdd (src_id, (dist, loc_estimate))*) – this is the result of the evaluator function

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

V

W