Touvlo Documentation

Benardi Nunes

Dec 30, 2020

Contents

Welcome to touvlo's documentation!

CHAPTER 1

Contents

1.1 Linear Regression

1.1.1 Single Parameter

class touvlo.lin_rg.**AbstractLinearRegression**(*theta=None*)

Represents the definitons common to all Linear Regressions.

Parameters **theta** (*numpy.array*) – Column vector of model's parameters. Defaults to None

theta

Column vector of model's parameters.

Type numpy.array

fit(*X*, *y*, *strategy='BGD'*, *alpha=0.1*, *num_iters=100*, ***kwargs*)

Adjusts model parameters to training data.

Parameters

- **X** (*numpy.array*) – Features' dataset.
- **y** (*numpy.array*) – Column vector of expected values.
- **strategy** (*str*) – Which optimization strategy should be employed:
  - 'BGD': Performs Batch Gradient Descent
  - 'SGD': Performs Stochastic Gradient Descent
  - 'MBGD': Performs Mini-Batch Gradient Descent
  - 'normal_equation': Employs Normal Equation
- **alpha** (*float*) – Learning rate or _step size of the optimization.
- **num_iters** (*int*) – Number of times the optimization will be performed.

Returns None

predict(*X*)

Computes prediction vector.

Parameters **X** (*numpy.array*) – Features' dataset.

Returns vector with a prediction for each example.

Return type numpy.array

class touvlo.lin_rg.**LinearRegression**(*theta=None*)

Represents an Unregularized Linear Regression model

Parameters **theta** (*numpy.array*) – Column vector of model's parameters. Defaults to None

theta

Column vector of model's parameters.

Type numpy.array

cost(*X*, *y*)

Computes the cost function J for Linear Regression.

Parameters

- **X** (*numpy.array*) – Features' dataset plus bias column.
- **y** (*numpy.array*) – Column vector of expected values.

Returns Computed cost.

Return type float

class touvlo.lin_rg.**RidgeLinearRegression**(*theta=None*, *_lambda=0*)

Represents a Ridge Linear Regression model

Parameters

- **theta** (*numpy.array*) – Column vector of model's parameters. Defaults to None
- **_lambda** (*float*) – The regularization hyperparameter.

theta

Column vector of model's parameters.

Type numpy.array

_lambda

The regularization hyperparameter.

Type float

cost(*X*, *y*, ***kwargs*)

Computes the regularized cost function J for Ridge Linear Regression.

Parameters

- **X** (*numpy.array*) – Features' dataset plus bias column.
- **y** (*numpy.array*) – Column vector of expected values.

Returns Computed cost.

Return type float

touvlo.lin_rg.**cost_func**(*X*, *y*, *theta*)

Computes the cost function J for Linear Regression.

Parameters

- **X** (*numpy.array*) – Features' dataset plus bias column.
- **y** (*numpy.array*) – Column vector of expected values.
- **theta** (*numpy.array*) – Column vector of model's parameters.

Returns Computed cost.

Return type float

`touvlo.lin_rg.grad`(*X*, *y*, *theta*)

Computes the gradient for Linear Regression.

Parameters

- **X** (*numpy.array*) – Features' dataset plus bias column.
- **y** (*numpy.array*) – Column vector of expected values.
- **theta** (*numpy.array*) – Column vector of model's parameters.

Returns Gradient column vector.

Return type numpy.array

`touvlo.lin_rg.h`(*X*, *theta*)

Linear regression hypothesis.

Parameters

- **X** (*numpy.array*) – Features' dataset plus bias column.
- **theta** (*numpy.array*) – Column vector of model's parameters.

Returns The projected value for each line of the dataset.

Return type numpy.array

`touvlo.lin_rg.normal_eqn`(*X*, *y*)

Produces optimal theta via normal equation.

Parameters

- **X** (*numpy.array*) – Features' dataset plus bias column.
- **y** (*numpy.array*) – Column vector of expected values.

Raises `LinAlgError`

Returns Optimized model parameters theta.

Return type numpy.array

`touvlo.lin_rg.predict`(*X*, *theta*)

Computes prediction vector.

Parameters

- **X** (*numpy.array*) – Features' dataset plus bias column.
- **theta** (*numpy.array*) – Column vector of model's parameters.

Returns vector with predictions for each input line.

Return type numpy.array

`touvlo.lin_rg.reg_cost_func`(*X*, *y*, *theta*, *_lambda*)

Computes the regularized cost function J for Linear Regression.

Parameters

- **X** (*numpy.array*) – Features' dataset plus bias column.
- **y** (*numpy.array*) – Column vector of expected values.
- **theta** (*numpy.array*) – Column vector of model's parameters.
- **_lambda** (*float*) – The regularization hyperparameter.

Returns Computed cost with regularization.

Return type float

touvlo.lin_rg.**reg_grad**(*X*, *y*, *theta*, *_lambda*)

Computes the regularized gradient for Linear Regression.

Parameters

- **X** (*numpy.array*) – Features' dataset plus bias column.
- **y** (*numpy.array*) – Column vector of expected values.
- **theta** (*numpy.array*) – Column vector of model's parameters.
- **_lambda** (*float*) – The regularization hyperparameter.

Returns Regularized gradient column vector.

Return type numpy.array

touvlo.lin_rg.**reg_normal_eqn**(*X*, *y*, *_lambda*)

Produces optimal theta via normal equation.

Parameters

- **X** (*numpy.array*) – Features' dataset plus bias column.
- **y** (*numpy.array*) – Column vector of expected values.
- **_lambda** (*float*) – The regularization hyperparameter.

Returns Optimized model parameters theta.

Return type numpy.array

1.2 Logistic Regression

1.2.1 Single Parameter

touvlo.lgx_rg.sgl_parm.**cost_func**(*X*, *Y*, *theta*)

Computes the cost function J for Logistic Regression.

Parameters

- **X** (*numpy.array*) – Features' dataset plus bias column.
- **Y** (*numpy.array*) – Column vector of expected values.
- **theta** (*numpy.array*) – Column vector of model's parameters.

Returns Computed cost.

Return type float

`touvlo.lgx_rg.sgl_parm.grad`(*X*, *Y*, *theta*)
Computes the gradient for the parameters theta.

Parameters

- **X** (*numpy.array*) – Features' dataset plus bias column.
- **Y** (*numpy.array*) – Column vector of expected values.
- **theta** (*numpy.array*) – Column vector of model's parameters.

Returns Gradient column vector.

Return type numpy.array

`touvlo.lgx_rg.sgl_parm.h`(*X*, *theta*)
Logistic regression hypothesis.

Parameters

- **X** (*numpy.array*) – Features' dataset plus bias column.
- **theta** (*numpy.array*) – Column vector of model's parameters.

Raises `ValueError`

Returns The probability that each entry belong to class 1.

Return type numpy.array

`touvlo.lgx_rg.sgl_parm.p`(*x*, *threshold=0.5*)
Predicts whether a probability falls into class 1.

Parameters

- **x** (*obj*) – Probability that example belongs to class 1.
- **threshold** (*float*) – point above which a probability is deemed of class 1.

Returns Binary value to denote class 1 or 0

Return type int

`touvlo.lgx_rg.sgl_parm.predict`(*X*, *theta*)
Classifies each entry as class 1 or class 0.

Parameters

- **X** (*numpy.array*) – Features' dataset plus bias column.
- **theta** (*numpy.array*) – Column vector of model's parameters.

Returns Column vector with each entry classification.

Return type numpy.array

`touvlo.lgx_rg.sgl_parm.predict_prob`(*X*, *theta*)
Produces the probability that the entries belong to class 1.

Returns Features' dataset plus bias column. theta (numpy.array): Column vector of model's parameters.

Return type X (numpy.array)

Raises `ValueError`

Returns The probability that each entry belong to class 1.

Return type numpy.array

`touvlo.lgx_rg.sgl_parm.`**`reg_cost_func`**(*X*, *Y*, *theta*, *_lambda*)

Computes the regularized cost function J for Logistic Regression.

Parameters

- **X** (*numpy.array*) – Features' dataset plus bias column.
- **Y** (*numpy.array*) – Column vector of expected values.
- **theta** (*numpy.array*) – Column vector of model's parameters.
- **_lambda** (*float*) – The regularization hyperparameter.

Returns Computed cost with regularization.

Return type float

`touvlo.lgx_rg.sgl_parm.`**`reg_grad`**(*X*, *Y*, *theta*, *_lambda*)

Computes the regularized gradient for Logistic Regression.

Parameters

- **X** (*numpy.array*) – Features' dataset plus bias column.
- **Y** (*numpy.array*) – Column vector of expected values.
- **theta** (*numpy.array*) – Column vector of model's parameters.
- **_lambda** (*float*) – The regularization hyperparameter.

Returns Regularized gradient column vector.

Return type numpy.array

1.2.2 Computation Graph

`touvlo.lgx_rg.cmpt_grf.`**`cost_func`**(*X*, *Y*, *hyp=None*, ***kwargs*)

Computes the cost function J for Logistic Regression.

Parameters

- **X** (*numpy.array*) – Features' dataset.
- **Y** (*numpy.array*) – Row vector of expected values.
- **hyp** (*numpy.array*) – The calculated model hypothesis, if not provided
- **named parameters to calculate it should be provided instead.** (*the*) –

Returns Computed cost.

Return type float

`touvlo.lgx_rg.cmpt_grf.`**`grad`**(*X*, *Y*, *w*, *b*)

Computes the gradient for the parameters w and b.

Parameters

- **X** (*numpy.array*) – Transpose features' dataset.
- **Y** (*numpy.array*) – Row vector of expected values.
- **w** (*numpy.array*) – Column vector of model's parameters.
- **b** (*float*) – Model's intercept parameter.

Returns

A 2-tuple consisting of a gradient column vector and a gradient value.

Return type (numpy.array, float)

`touvlo.lgx_rg.cmpt_grf.h`(*X*, *w*, *b*)

Logistic regression hypothesis.

Parameters

- **X** (*numpy.array*) – Transposed features' dataset.
- **w** (*numpy.array*) – Column vector of model's parameters.
- **b** (*float*) – Model's intercept parameter.

Returns The probability that each entry belong to class 1.

Return type numpy.array

`touvlo.lgx_rg.cmpt_grf.predict`(*X*, *w*, *b*, *threshold=0.5*)

Predicts whether the given probabilities fall into class 1.

Parameters

- **X** (*numpy.array*) – Transpose features' dataset.
- **threshold** (*float*) – Point above which a probability is assigned
- **class 1.** (*to*) –

Returns Binary value to denote class 1 or 0 for each example.

Return type numpy.array

1.3 Classification Neural Network

1.3.1 Single Parameter

`touvlo.nnet.sgl_parm.back_propagation`(*y*, *theta*, *a*, *z*, *num_labels*, *n_hidden_layers=1*)

Applies back propagation to minimize model's loss.

Parameters

- **y** (*numpy.array*) – Column vector of expected values.
- **theta** (*numpy.array (numpy.array)*) – array of model's weight matrices by layer.
- **a** (*numpy.array (numpy.array)*) – array of activation matrices by layer.
- **z** (*numpy.array (numpy.array)*) – array of parameters prior to sigmoid by layer.
- **num_labels** (*int*) – Number of classes in multiclass classification.
- **n_hidden_layers** (*int*) – Number of hidden layers in network.

Returns array of matrices of 'error values' by layer.

Return type numpy.array(numpy.array)

`touvlo.nnet.sgl_parm.cost_function`(*X*, *y*, *theta*, *_lambda*, *num_labels*, *n_hidden_layers=1*)

Computes the cost function J for Neural Network.

Parameters

- **X** (*numpy.array*) – Features' dataset.

- **y** (*numpy.array*) – Column vector of expected values.
- **theta** (*numpy.array*) – Column vector of model's parameters.
- **_lambda** (*float*) – The regularization hyperparameter.
- **num_labels** (*int*) – Number of classes in multiclass classification.
- **n_hidden_layers** (*int*) – Number of hidden layers in network.

Returns Computed cost.

Return type float

touvlo.nnet.sgl_parm.**feed_forward**(*X*, *theta*, *n_hidden_layers=1*)

Applies forward propagation to calculate model's hypothesis.

Parameters

- **X** (*numpy.array*) – Features' dataset.
- **theta** (*numpy.array*) – Column vector of model's parameters.
- **n_hidden_layers** (*int*) – Number of hidden layers in network.

Returns A 2-tuple consisting of an array of parameters prior to activation by layer and an array of activation matrices by layer.

Return type (numpy.array(numpy.array), numpy.array(numpy.array))

touvlo.nnet.sgl_parm.**grad**(*X*, *y*, *nn_params*, *_lambda*, *input_layer_size*, *hidden_layer_size*, *num_labels*, *n_hidden_layers=1*)

Calculates gradient of neural network's parameters.

Parameters

- **X** (*numpy.array*) – Features' dataset.
- **y** (*numpy.array*) – Column vector of expected values.
- **nn_params** (*numpy.array*) – Column vector of model's parameters.
- **_lambda** (*float*) – The regularization hyperparameter.
- **input_layer_size** (*int*) – Number of units in the input layer.
- **hidden_layer_size** (*int*) – Number of units in a hidden layer.
- **num_labels** (*int*) – Number of classes in multiclass classification.
- **n_hidden_layers** (*int*) – Number of hidden layers in network.

Returns array of gradient values by weight matrix.

Return type numpy.array(numpy.array)

touvlo.nnet.sgl_parm.**h**(*X*, *theta*, *n_hidden_layers=1*)

Classification Neural Network hypothesis.

Parameters

- **X** (*numpy.array*) – Features' dataset.
- **theta** (*numpy.array*) – Column vector of model's parameters.
- **n_hidden_layers** (*int*) – Number of hidden layers in network.

Returns The probability that each entry belong to class 1.

Return type numpy.array

`touvlo.nnet.sgl_parm.`**`init_nn_weights`**(*input_layer_size*, *hidden_layer_size*, *num_labels*, *n_hidden_layers=1*)

Initialize the weight matrices of a network with random values.

Parameters

- **hidden_layer_size** (*int*) – Number of units in a hidden layer.
- **input_layer_size** (*int*) – Number of units in the input layer.
- **num_labels** (*int*) – Number of classes in multiclass classification.
- **n_hidden_layers** (*int*) – Number of hidden layers in network.

Returns array of weight matrices of random values.

Return type numpy.array(numpy.array)

`touvlo.nnet.sgl_parm.`**`rand_init_weights`**(*L_in*, *L_out*)

Initializes weight matrix with random values.

Parameters

- **X** (*numpy.array*) – Features' dataset.
- **L_in** (*int*) – Number of units in previous layer.
- **n_hidden_layers** (*int*) – Number of units in next layer.

Returns Random values' matrix of conforming dimensions.

Return type numpy.array

`touvlo.nnet.sgl_parm.`**`unravel_params`**(*nn_params*, *input_layer_size*, *hidden_layer_size*, *num_labels*, *n_hidden_layers=1*)

Unravels flattened array into list of weight matrices

Parameters

- **nn_params** (*numpy.array*) – Row vector of model's parameters.
- **input_layer_size** (*int*) – Number of units in the input layer.
- **hidden_layer_size** (*int*) – Number of units in a hidden layer.
- **num_labels** (*int*) – Number of classes in multiclass classification.
- **n_hidden_layers** (*int*) – Number of hidden layers in network.

Returns array with model's weight matrices.

Return type numpy.array(numpy.array)

1.3.2 Computation Graph

`touvlo.nnet.cmpt_grf.`**`L_model_backward`**(*AL*, *Y*, *caches*)

Implements the backward propagation for the [LINEAR->RELU] * (L-1) -> LINEAR -> SIGMOID group.

Parameters

- **AL** (*numpy.array*) – probability vector, output of the forward propagation (L_model_forward()).
- **Y** (*numpy.array*) – true "label" vector (containing 0 if non-cat, 1 if cat)

- **caches** (*list(tuple)*) – list of caches containing every cache of linear_activation_forward() with "relu" (it's caches[l], for l in range(L-1) i.e l = 0…L-2) the cache of linear_activation_forward() with "sigmoid" (it's caches[L-1]).

Returns

A dictionary with the gradients:

- grads["dA" + str(l)] = …
- grads["dW" + str(l)] = …
- grads["db" + str(l)] = …

Return type (dict)

`touvlo.nnet.cmpt_grf.`**`L_model_forward`**(*X*, *parameters*)

Implements forward propagation for the [LINEAR->RELU] * (L-1) -> LINEAR -> SIGMOID computation.

Parameters

- **X** (*numpy.array) data of shape (input size, number of examples*) –
- **parameters** (*dict) output of initialize_parameters_deep()* –

Returns A 2-tuple consisting of the last post-activation value and a list of caches containing every cache of linear_activation_forward(), (there are L-1 of them, indexed from 0 to L-1).

Return type (numpy.array, list[tuple])

`touvlo.nnet.cmpt_grf.`**`compute_cost`**(*AL*, *Y*)

Implements the cost function.

Parameters

- **AL** (*numpy.array*) – probability vector corresponding to your label predictions, shape (1, number of examples)
- **Y** (*numpy.array*) – true "label" vector (for example: containing 0 if non-cat, 1 if cat), shape (1, number of examples)

Returns cross-entropy cost

Return type float

`touvlo.nnet.cmpt_grf.`**`init_params`**(*layer_dims*, *_seed=1*)

Creates numpy arrays to to represent the weight matrices and intercepts of the Neural Network.

Parameters

- **layer_dims** (*list[int]*) – List of numbers representing the dimensions of each layer in our network.
- **_seed** (*int*) – Seed to make function reproducible despite randomness.

Returns Single dictionary containing your parameters "W1", "b1", …, "WL", "bL" where Wl is a weight matrix of shape (layer_dims[l], layer_dims[l-1]) and bl is the bias vector of shape (layer_dims[l], 1).

Return type dict

`touvlo.nnet.cmpt_grf.`**`linear_activation_backward`**(*dA*, *cache*, *activation*)

Implements the backward propagation for the LINEAR -> ACTIVATION layer.

Parameters

- **dA** (*numpy.array*) – post-activation gradient for current layer l.
- **cache** (*tuple*) – values (linear_cache, activation_cache) we store for computing backward propagation efficiently
- **activation** (*str*) – the activation to be used in this layer, stored as a text string: "sigmoid" or "relu".

Returns A 3-tuple consisting of the Gradient of the cost with respect to the activation (of the previous layer l-1), same shape as A_prev, the Gradient of the cost with respect to W (current layer l), same shape as W and the Gradient of the cost with respect to b (current layer l), same shape as b.

Return type (numpy.array, numpy.array, float)

touvlo.nnet.cmpt_grf.**linear_activation_forward**(*A_prev*, *W*, *b*, *activation*)

Implement the forward propagation for the LINEAR->ACTIVATION layer

Parameters

- **A_prev** (*numpy.array*) – activations from previous layer (or input data): (size of previous layer, number of examples)
- **W** (*numpy.array*) – numpy array of shape (size of current layer, size of previous layer)
- **b** (*float*) – bias vector, numpy array of shape (size of the current layer, 1)
- **activation** (*str*) – the activation to be used in this layer, stored as a text string: "sigmoid" or "relu"

Returns A 2-tuple consisting of the output of the activation function, also called the post-activation value and a python tuple containing "linear_cache" and "activation_cache"; stored for computing the backward pass efficiently.

Return type (numpy.array, dict)

touvlo.nnet.cmpt_grf.**linear_backward**(*dZ*, *cache*)

Implements the linear portion of backward propagation for a single layer (layer l).

Parameters

- **dZ** (*numpy.array*) – Gradient of the cost with respect to the linear output (of current layer l).
- **cache** (*tuple*) – values (A_prev, W, b) coming from the forward propagation in the current layer.

Returns A 3-tuple consisting of the Gradient of the cost with respect to the activation (of the previous layer l-1), same shape as A_prev, the Gradient of the cost with respect to W (current layer l),same shape as W and the Gradient of the cost with respect to b (current layer l), same shape as b.

Return type (numpy.array, numpy.array, float)

touvlo.nnet.cmpt_grf.**linear_forward**(*A*, *W*, *b*)

Implement the linear part of a layer's forward propagation.

Parameters

- **A** (*numpy.array*) – activations from previous layer (or input data): (size of previous layer, number of examples).
- **W** (*numpy.array*) – weights matrix: numpy array of shape (size of current layer, size of previous layer).

- **b** (*numpy.array*) – bias vector, numpy array of shape (size of the current layer, 1).

Returns A 2-tuple consisting of the input of the activation function, also called pre-activation parameter and a python tuple containing "A", "W" and "b" ; stored for computing the backward pass efficiently.

Return type (numpy.array, dict)

`touvlo.nnet.cmpt_grf.update_parameters`(*parameters*, *grads*, *learning_rate*)

Updates parameters using gradient descent.

Parameters

- **parameters** (*dict*) – dictionary containing your parameters
- **grads** (*dict*) – dictionary containing your gradients, output of L_model_backward

Returns

(dict) dictionary containing your updated parameters

- parameters["W" + str(l)] = . . .
- parameters["b" + str(l)] = . . .

1.4 Recommender Systems

1.4.1 Collaborative Filtering

`touvlo.rec_sys.cf.cost_function`(*X*, *Y*, *R*, *theta*, *_lambda*)

Computes the cost function J for Collaborative Filtering.

Parameters

- **X** (*numpy.array*) – Matrix of product features.
- **Y** (*numpy.array*) – Scores' matrix.
- **R** (*numpy.array*) – Matrix of 0s and 1s (whether there's a rating).
- **theta** (*numpy.array*) – Matrix of user features.
- **_lambda** (*float*) – The regularization hyperparameter.

Returns Computed cost.

Return type float

`touvlo.rec_sys.cf.grad`(*params*, *Y*, *R*, *num_users*, *num_products*, *num_features*, *_lambda*)

Calculates gradient of Collaborative Filtering's parameters

Parameters

- **params** (*numpy.array*) – flattened product and user features..
- **Y** (*numpy.array*) – Scores' matrix.
- **R** (*numpy.array*) – Matrix of 0s and 1s (whether there's a rating).
- **num_users** (*int*) – Number of users in this instance.
- **num_products** (*int*) – Number of products in this instance.
- **num_features** (*int*) – Number of features in this instance.

- **_lambda** (*float*) – The regularization hyperparameter.

Returns Flattened gradient of product and user parameters.

Return type numpy.array

`touvlo.rec_sys.cf.`**`unravel_params`**(*params*, *num_users*, *num_products*, *num_features*)
Unravels flattened array into features' matrices

Parameters

- **params** (*numpy.array*) – Row vector of coefficients.
- **num_users** (*int*) – Number of users in this instance.
- **num_products** (*int*) – Number of products in this instance.
- **num_features** (*int*) – Number of features in this instance.

Returns A 2-tuple consisting of a matrix of product features and a matrix of user features.

Return type (numpy.array, numpy.array)

1.5 Unsupervised learning

1.5.1 PCA

`touvlo.unsupv.pca.`**`pca`**(*X*)
Runs Principal Component Analysis on dataset

Parameters **X** (*numpy.array*) – Features' dataset

Returns

A 2-tuple of U, eigenvectors of covariance matrix, and S, eigenvalues (on diagonal) of covariance matrix.

Return type (numpy.array, numpy.array)

`touvlo.unsupv.pca.`**`project_data`**(*X*, *U*, *k*)
Computes reduced data representation (projected data)

Parameters

- **X** (*numpy.array*) – Normalized features' dataset
- **U** (*numpy.array*) – eigenvectors of covariance matrix
- **k** (*int*) – Number of features in reduced data representation

Returns Reduced data representation (projection)

Return type numpy.array

`touvlo.unsupv.pca.`**`recover_data`**(*Z*, *U*, *k*)
Recovers an approximation of original data using the projected data

Parameters

- **Z** (*numpy.array*) – Reduced data representation (projection)
- **U** (*numpy.array*) – eigenvectors of covariance matrix
- **k** (*int*) – Number of features in reduced data representation

Returns Approximated features' dataset

Return type numpy.array

1.5.2 K-means

`touvlo.unsupv.kmeans.`**`compute_centroids`**(*X*, *idx*, *K*)

Computes centroids from the mean of its cluster's members.

Computes centroids from the mean of its cluster's members if there are any members for the centroid, else it returns an array of nan.

Parameters

- **X** (*numpy.array*) – Features' dataset
- **idx** (*numpy.array*) – Column vector of assigned centroids' indices.
- **K** (*int*) – Number of centroids.

Returns Column vector of newly computed centroids

Return type numpy.array

`touvlo.unsupv.kmeans.`**`cost_function`**(*X*, *idx*, *centroids*)

Calculates the cost function for K means.

Parameters

- **X** (*numpy.array*) – Features' dataset
- **idx** (*numpy.array*) – Column vector of assigned centroids' indices.

Returns Computed cost

Return type float

`touvlo.unsupv.kmeans.`**`elbow_method`**(*X*, *K_values*, *max_iters*, *n_inits*)

Calculates the cost for each given K.

Parameters

- **X** (*numpy.array*) – Features' dataset
- **K_values** (*list (int)*) – List of possible number of centroids.
- **max_iters** (*int*) – Number of times the algorithm will be fitted.
- **n_inits** (*int*) – Number of random initialization.

Returns a list of cost values for each K.

Return type (list(float))

`touvlo.unsupv.kmeans.`**`euclidean_dist`**(*p*, *q*)

Calculates Euclidean distance between 2 n-dimensional points.

Parameters

- **p** (*numpy.array*) – First n-dimensional point.
- **q** (*numpy.array*) – Second n-dimensional point.

Returns Distance between 2 points.

Return type float

`touvlo.unsupv.kmeans.`**`find_closest_centroids`**(*X*, *initial_centroids*)

Assigns to each example the indice of the closest centroid.

Parameters

- **X** (*numpy.array*) – Features' dataset
- **initial_centroids** (*numpy.array*) – List of initialized centroids.

Returns Column vector of assigned centroids' indices.

Return type numpy.array

`touvlo.unsupv.kmeans.`**`init_centroids`**(*X*, *K*)

Computes centroids from the mean of its cluster's members.

Parameters

- **X** (*numpy.array*) – Features' dataset
- **idx** (*numpy.array*) – Column vector of assigned centroids' indices.
- **K** (*int*) – Number of centroids.

Returns Column vector of centroids randomly picked from dataset

Return type numpy.array

`touvlo.unsupv.kmeans.`**`run_intensive_kmeans`**(*X*, *K*, *max_iters*, *n_inits*)

Applies kmeans using multiple random initializations.

Parameters

- **X** (*numpy.array*) – Features' dataset
- **K** (*int*) – Number of centroids.
- **max_iters** (*int*) – Number of times the algorithm will be fitted.
- **n_inits** (*int*) – Number of random initialization.

Returns

A 2-tuple of centroids, a column vector of centroids, and idx, a column vector of assigned centroids' indices.

Return type (numpy.array, numpy.array)

`touvlo.unsupv.kmeans.`**`run_kmeans`**(*X*, *K*, *max_iters*)

Applies kmeans using a single random initialization.

Parameters

- **X** (*numpy.array*) – Features' dataset
- **K** (*int*) – Number of centroids.
- **max_iters** (*int*) – Number of times the algorithm will be fitted.

Returns

A 2-tuple of centroids, a column vector of centroids, and idx, a column vector of assigned centroids' indices.

Return type (numpy.array, numpy.array)

1.5.3 Anomaly Detection

`touvlo.unsupv.anmly_detc.`**`cov_matrix`**(*X*, *mu*)

Calculates the covariance matrix for matrix X (m x n).

Parameters

- **X** (*numpy.array*) – Features' dataset.
- **mu** (*numpy.array*) – Mean of each feature/column of.

Returns Covariance matrix (n x n)

Return type int

`touvlo.unsupv.anmly_detc.`**`estimate_multi_gaussian`**(*X*)

Estimates parameters for Multivariate Gaussian distribution.

Parameters **X** (*numpy.array*) – Features' dataset.

Returns

A 2-tuple of mu, the mean of each feature/column of X, and sigma, the covariance matrix for X.

Return type (numpy.array, numpy.array)

`touvlo.unsupv.anmly_detc.`**`estimate_uni_gaussian`**(*X*)

Estimates parameters for Univariate Gaussian distribution.

Parameters **X** (*numpy.array*) – Features' dataset.

Returns

A 2-tuple of mu, the mean of each feature/column of X, and sigma2, the variance of each feature/column of X.

Return type (numpy.array, numpy.array)

`touvlo.unsupv.anmly_detc.`**`is_anomaly`**(*p*, *threshold=0.5*)

Predicts whether a probability falls into class 1 (anomaly).

Parameters

- **p** (*numpy.array*) – Probability that example belongs to class 1 (is anomaly).
- **threshold** (*float*) – point below which an example is considered of class 1.

Returns Binary value to denote class 1 or 0

Return type int

`touvlo.unsupv.anmly_detc.`**`multi_gaussian`**(*X*, *mu*, *sigma*)

Estimates probability that examples belong to Multivariate Gaussian.

Parameters

- **X** (*numpy.array*) – Features' dataset.
- **mu** (*numpy.array*) – Mean of each feature/column of X.
- **sigma** (*numpy.array*) – Covariance matrix for X.

Returns Probability density function for each example

Return type numpy.array

`touvlo.unsupv.anmly_detc.`**`predict`**(*X*, *epsilon*, *gaussian*, ***kwargs*)

Predicts whether examples are anomalies.

Parameters

- **X** (*numpy.array*) – Features' dataset.
- **epsilon** (*float*) – point below which an example is considered of class 1.
- **gaussian** (*numpy.array*) – Function that estimates pertinency probability.

Returns Column vector of classification

Return type numpy.array

`touvlo.unsupv.anmly_detc.`**`uni_gaussian`**(*X*, *mu*, *sigma2*)

Estimates probability that examples belong to Univariate Gaussian.

Parameters

- **X** (*numpy.array*) – Features' dataset.
- **mu** (*numpy.array*) – Mean of each feature/column of X.
- **sigma2** (*numpy.array*) – Variance of each feature/column of X.

Returns Probability density function for each example

Return type numpy.array

1.6 Utils

`touvlo.utils.`**`BGD`**(*X*, *y*, *grad*, *initial_theta*, *alpha*, *num_iters*, ***kwargs*)

Performs parameter optimization via Batch Gradient Descent.

Parameters

- **X** (*numpy.array*) – Features' dataset plus bias column.
- **y** (*numpy.array*) – Column vector of expected values.
- **grad** (*numpy.array*) – Routine that generates the partial derivatives given theta.
- **initial_theta** (*numpy.array*) – Initial value for parameters to be optimized.
- **alpha** (*float*) – Learning rate or _step size of the optimization.
- **num_iters** (*int*) – Number of times the optimization will be performed.

Returns Optimized model parameters.

Return type numpy.array

`touvlo.utils.`**`MBGD`**(*X*, *y*, *grad*, *initial_theta*, *alpha*, *num_iters*, *b*, ***kwargs*)

Performs parameter optimization via Mini-Batch Gradient Descent.

Parameters

- **X** (*numpy.array*) – Features' dataset plus bias column.
- **y** (*numpy.array*) – Column vector of expected values.
- **grad** (*numpy.array*) – Routine that generates the partial derivatives given theta.
- **initial_theta** (*numpy.array*) – Initial value for parameters to be optimized.
- **alpha** (*float*) – Learning rate or _step size of the optimization.

- **num_iters** (*int*) – Number of times the optimization will be performed.
- **b** (*int*) – Number of examples in mini batch.

Returns Optimized model parameters.

Return type numpy.array

touvlo.utils.**SGD**(*X*, *y*, *grad*, *initial_theta*, *alpha*, *num_iters*, ***kwargs*)

Performs parameter optimization via Stochastic Gradient Descent.

Parameters

- **X** (*numpy.array*) – Features' dataset plus bias column.
- **y** (*numpy.array*) – Column vector of expected values.
- **grad** (*numpy.array*) – Routine that generates the partial derivatives given theta.
- **initial_theta** (*numpy.array*) – Initial value for parameters to be optimized.
- **alpha** (*float*) – Learning rate or _step size of the optimization.
- **num_iters** (*int*) – Number of times the optimization will be performed.

Returns Optimized model parameters.

Return type numpy.array

touvlo.utils.**feature_normalize**(*X*)

Performs Z score normalization in a numeric dataset.

Parameters **X** (*numpy.array*) – Features' dataset plus bias column.

Returns

A 3-tuple of X_norm, normalized features' dataset, mu, mean of each feature, and sigma, standard deviation of each feature.

Return type (numpy.array, numpy.array, numpy.array)

touvlo.utils.**g**(*x*)

This function applies the sigmoid function on a given value.

Parameters **x** (*obj*) – Input value or object containing value .

Returns Sigmoid function at value.

Return type obj

touvlo.utils.**g_grad**(*x*)

This function calculates the sigmoid gradient at a given value.

Parameters **x** (*obj*) – Input value or object containing value .

Returns Sigmoid gradient at value.

Return type obj

touvlo.utils.**mean_normlztn**(*Y*, *R*)

Performs mean normalization in a numeric dataset.

Parameters

- **Y** (*numpy.array*) – Scores' dataset.
- **R** (*numpy.array*) – Dataset of 0s and 1s (whether there's a rating).

Returns

- Y_norm - Normalized scores' dataset (row wise).
- Y_mean - Column vector of calculated means.

Return type

- Y_norm (:py:class: numpy.array)
- Y_mean (:py:class: numpy.array)

`touvlo.utils.`**`numerical_grad`**(*J*, *theta*, *err*)

Numerically calculates the gradient of a given cost function.

Parameters

- **J** (*Callable*) – Function handle that computes cost given theta.
- **theta** (*numpy.array*) – Model parameters.
- **err** (*float*) – distance between points where J is evaluated.

Returns Computed numeric gradient.

Return type numpy.array

`touvlo.utils.`**`relu`**(*Z*)

Implement the RELU function.

Arguments: Z – Output of the linear layer, of any shape

Returns: A – Post-activation parameter, of the same shape as Z cache – a python dictionary containing "A" ; stored for

computing the backward pass efficiently

`touvlo.utils.`**`relu_backward`**(*dA*, *cache*)

Implement the backward propagation for a single RELU unit.

Arguments: dA – post-activation gradient, of any shape cache – 'Z' where we store for computing backward propagation efficiently

Returns: dZ – Gradient of the cost with respect to Z

`touvlo.utils.`**`sigmoid`**(*Z*)

Implements the sigmoid activation in numpy

Arguments: Z – numpy array of any shape

Returns: A – output of sigmoid(z), same shape as Z cache – returns Z as well, useful during backpropagation

`touvlo.utils.`**`sigmoid_backward`**(*dA*, *cache*)

Implement the backward propagation for a single SIGMOID unit.

Arguments: dA – post-activation gradient, of any shape cache – 'Z' where we store for computing backward propagation efficiently

Returns: dZ – Gradient of the cost with respect to Z

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index

Symbols

A

B

C

E

F

G

H

I

K

L

M

N

P

R

S

T

U