DragoNN Documentation

Release 0.1

Johnny Israeli, Michael Wainberg, Avanti Shrikumar

June 15, 2016

CONTENTS

Contents:

CHAPTER

ONE

DRAGONN PACKAGE

1.1 Subpackages

1.1.1 dragonn.synthetic package

Submodules

dragonn.synthetic.synthetic module

class dragonn.synthetic.synthetic.**AbstractApplySingleMutationFromSet**(*setOfMutations*, *name=None*)

Bases: *dragonn.synthetic.synthetic.AbstractTransformation*

Class for applying a single mutation from a set of mutations; used to transform substrings generated by another method

getClassName()

getJsonableObject()

selectMutation()

transform(*stringArr*)

class dragonn.synthetic.synthetic.**AbstractBackgroundGenerator**

Bases: object

Returns the sequence that the embeddings are subsequently inserted into.

generateBackground()

getJsonableObject()

class dragonn.synthetic.synthetic.**AbstractEmbeddable**

Bases: object

Represents a thing which can be embedded. Note that an Embeddable + a position = an embedding.

canEmbed(*priorEmbeddedThings*, *startPos*)

priorEmbeddedThings: instance of AbstractPriorEmbeddedThings startPos: the position you are considering embedding self at returns a boolean indicating whether self can be embedded at startPos,

given the things that have already been embedded.

embedInBackgroundStringArr(*priorEmbeddedThings*, *backgroundStringArr*, *startPos*)
Will embed self at startPos in backgroundStringArr, and will update priorEmbeddedThings. priorEmbeddedThings: instance of AbstractPriorEmbeddedThings backgroundStringArr: an array of characters representing the background startPos: the position to embed self at

getDescription()

class dragonn.synthetic.synthetic.**AbstractEmbeddableGenerator**(*name*)
Bases: *dragonn.synthetic.synthetic.DefaultNameMixin*

Generates an embeddable, usually for embedding in a background sequence.

generateEmbeddable()

getJsonableObject()

class dragonn.synthetic.synthetic.**AbstractEmbedder**(*name*)
Bases: *dragonn.synthetic.synthetic.DefaultNameMixin*

class that is used to embed things in a sequence

embed(*backgroundStringArr*, *priorEmbeddedThings*, *additionalInfo=None*)
backgroundStringArr: array of characters representing the background string priorEmbeddedThings: instance of AbstractPriorEmbeddedThings. additionalInfo: instance of AdditionalInfo; allows the embedder to send back info about what it did modifies: backgroundStringArr to include whatever this class has embedded

getJsonableObject()

class dragonn.synthetic.synthetic.**AbstractLoadedMotifs**(*fileName*, *pseudoCountProb=0.0*, *background=OrderedDict([('A', 0.27), ('C', 0.23), ('G', 0.23), ('T', 0.27)])*)
Bases: object

A class that contains instances of pwm.PWM loaded from a file. The pwms can be accessed by name.

getJsonableObject()

getPwm(*name*)
returns the pwm.PWM instance with the specified name.

getReadPwmAction(*recordedPwms*)
This is the action that is to be performed on each line of the file when it is read in. recordedPwms is an OrderedDict that stores instances of pwm.PWM

class dragonn.synthetic.synthetic.**AbstractPositionGenerator**(*name*)
Bases: *dragonn.synthetic.synthetic.DefaultNameMixin*

Given the length of the background sequence and the length of the substring you are trying to embed, will return a start position to embed the substring at.

generatePos(*lenBackground*, *lenSubstring*, *additionalInfo=None*)

getJsonableObject()

class dragonn.synthetic.synthetic.**AbstractPriorEmbeddedThings**
Bases: object

class that is used to keep track of what has already been embedded in a sequence

addEmbedding(*startPos*, *what*)
embeds "what" from startPos to startPos+len(what). Creates an Embedding object

canEmbed(*startPos*, *endPos*)
returns a boolean indicating whether the region from startPos to endPos is available for embedding

getEmbeddings()
returns a collection of Embedding objects

getNumOccupiedPos()
returns the number of posiitons that are filled with some kind of embedding

getTotalPos()
returns the total number of positions available to embed things in

class dragonn.synthetic.synthetic.**AbstractQuantityGenerator**(*name*)
Bases: *dragonn.synthetic.synthetic.DefaultNameMixin*

class to sample according to a distribution

generateQuantity()
returns the sampled value

getJsonableObject()

class dragonn.synthetic.synthetic.**AbstractSequenceSetGenerator**
Bases: object

class that is used to return a generator for a collection of generated sequences.

generateSequences()
returns a generator of GeneratedSequence objects

getJsonableObject()
returns an object representing the details of this, which can be converted to json.

class dragonn.synthetic.synthetic.**AbstractSetOfMutations**(*mutationsArr*)
Bases: object

Represents a collection of pwm.Mutation objects

getJsonableObject()

getMutationsArr()

class dragonn.synthetic.synthetic.**AbstractSingleSequenceGenerator**(*namePrefix=None*)
Bases: object

When called, generates a single sequence

generateSequence()
returns GeneratedSequence object

getJsonableObject()
returns an object representing the details of this, which can be converted to json.

class dragonn.synthetic.synthetic.**AbstractSubstringGenerator**(*name*)
Bases: *dragonn.synthetic.synthetic.DefaultNameMixin*

Generates a substring, usually for embedding in a background sequence.

generateSubstring()

getJsonableObject()

class dragonn.synthetic.synthetic.**AbstractTransformation**(*name*)
Bases: *dragonn.synthetic.synthetic.DefaultNameMixin*

takes an array of characters, applies some transformation, returns an array of characters (may be the same (mutated) one or a different one)

getJsonableObject()

transform(*stringArr*)
stringArr is an array of characters. Returns an array of characters that has the transformation applied. May mutate stringArr

class dragonn.synthetic.synthetic.**AdditionalInfo**
Bases: object

isInTrace(*operatorName*)

updateAdditionalInfo(*operatorName*, *value*)

updateTrace(*operatorName*)

class dragonn.synthetic.synthetic.**AllEmbedders**(*embedders*, *name=None*)
Bases: *dragonn.synthetic.synthetic.AbstractEmbedder*

Wrapper around a list of embedders to make sure all are called Useful in conjunciton with RandomSubsetOfEmbedders

getJsonableObject()

class dragonn.synthetic.synthetic.**BernoulliQuantityGenerator**(*prob*, *name=None*)
Bases: *dragonn.synthetic.synthetic.AbstractQuantityGenerator*

Generates 1 or 0 according to a bernoulli distribution

generateQuantity()

getJsonableObject()

class dragonn.synthetic.synthetic.**BestHitPwm**(*pwm*, *bestHitMode='pwmProb'*, *name=None*)
Bases: *dragonn.synthetic.synthetic.AbstractSubstringGenerator*

always returns the best possible match to the pwm in question when called

generateSubstring()

getJsonableObject()

class dragonn.synthetic.synthetic.**BestHitPwmFromLoadedMotifs**(*loadedMotifs*, *motifName*, *bestHitMode='pwmProb'*, *name=None*)
Bases: *dragonn.synthetic.synthetic.BestHitPwm*

convenience wrapper class for instantiating parent by pulling the pwm given the name from an AbstractLoadedMotifs object (it basically extracts the pwm for you)

getJsonableObject()

class dragonn.synthetic.synthetic.**ChooseMutationAtRandom**(*setOfMutations*, *name=None*)
Bases: *dragonn.synthetic.synthetic.AbstractApplySingleMutationFromSet*

Selects a mutation at random from self.setOfMutations to apply; see parent docs.

getClassName()

selectMutation()

class dragonn.synthetic.synthetic.**ChooseValueFromASet**(*setOfPossibleValues*, *name=None*)
Bases: *dragonn.synthetic.synthetic.AbstractQuantityGenerator*

Randomly samples a particular value from a set of values

generateQuantity()

getJsonableObject()

class dragonn.synthetic.synthetic.**DefaultNameMixin**(*name*)
Bases: object

getDefaultName()

class dragonn.synthetic.synthetic.**EmbedInABackground**(*backgroundGenerator*, *embedders*, *namePrefix=None*)
Bases: *dragonn.synthetic.synthetic.AbstractSingleSequenceGenerator*

Takes a backgroundGenerator and a series of embedders. Will generate the background and then call each of the embedders in succession. Then returns the result.

generateSequence()
generates a background using self.backgroundGenerator, splits it into an array, and passes it to each of self.embedders in turn for embedding things. returns an instance of GeneratedSequence

getJsonableObject()
see parent

class dragonn.synthetic.synthetic.**EmbeddableEmbedder**(*embeddableGenerator*, *positionGenerator=<dragonn.synthetic.synthetic.UniformPositionGenerator object>*, *name=None*)
Bases: *dragonn.synthetic.synthetic.AbstractEmbedder*

Embeds instances of AbstractEmbeddable within the background sequence, at a position sampled from a distribution. Only embeds at unoccupied positions

getJsonableObject()

class dragonn.synthetic.synthetic.**Embedding**(*what*, *startPos*)
Bases: object

Represents something that has been embedded in a sequence

classmethod fromString(*string*, *whatClass=None*)

class dragonn.synthetic.synthetic.**FixedQuantityGenerator**(*quantity*, *name=None*)
Bases: *dragonn.synthetic.synthetic.AbstractQuantityGenerator*

returns a fixed number every time generateQuantity is called

generateQuantity()

getJsonableObject()

class dragonn.synthetic.synthetic.**FixedSubstringGenerator**(*fixedSubstring*, *name=None*)
Bases: *dragonn.synthetic.synthetic.AbstractSubstringGenerator*

When generateSubstring() is called, always returns the same string. The string also serves as its own description

generateSubstring()

getJsonableObject()

class dragonn.synthetic.synthetic.**GenerateSequenceNTimes**(*singleSetGenerator*, *N*)
Bases: *dragonn.synthetic.synthetic.AbstractSequenceSetGenerator*

If you just want to use a generator of a single sequence and call it N times, use this class.

generateSequences()
calls singleSetGenerator N times.

getJsonableObject()

class dragonn.synthetic.synthetic.**GeneratedSequence**(*seqName*, *seq*, *embeddings*, *additionalInfo*)
Bases: object

An object representing a sequence that has been generated.

class dragonn.synthetic.synthetic.**InsideCentralBp**(*centralBp*, *name=None*)
Bases: *dragonn.synthetic.synthetic.AbstractPositionGenerator*

returns a position within the central region of a background sequence, sampled uniformly at random

getJsonableObject()

class dragonn.synthetic.synthetic.**IsInTraceLabelGenerator**(*labelNames*)
Bases: *dragonn.synthetic.synthetic.LabelGenerator*

class dragonn.synthetic.synthetic.**LabelGenerator**(*labelNames*, *labelsFromGeneratedSequenceFunction*)
Bases: object

generateLabels(*generatedSequence*)

class dragonn.synthetic.synthetic.**LoadedEncodeMotifs**(*fileName*, *pseudocountProb=0.0*, *background=OrderedDict([('A', 0.27), ('C', 0.23), ('G', 0.23), ('T', 0.27)])*)
Bases: *dragonn.synthetic.synthetic.AbstractLoadedMotifs*

This class is specifically for reading files in the encode motif format - specifically the motifs.txt file that contains Pouya's motifs

getReadPwmAction(*recordedPwms*)

class dragonn.synthetic.synthetic.**MinMaxWrapper**(*quantityGenerator*, *theMin=None*, *theMax=None*, *name=None*)
Bases: *dragonn.synthetic.synthetic.AbstractQuantityGenerator*

Wrapper that restricts a distribution to only return values between the min and the max. If a value outside the range is returned, resamples until it obtains a value within the range. Warns if it resamples too many times.

generateQuantity()

getJsonableObject()

class dragonn.synthetic.synthetic.**OutsideCentralBp**(*centralBp*, *name=None*)
Bases: *dragonn.synthetic.synthetic.AbstractPositionGenerator*

Returns a position OUTSIDE the central region of a background sequence, sampled uniformly at random. Complement of InsideCentralBp.

getJsonableObject()

class dragonn.synthetic.synthetic.**PairEmbeddable**(*string1*, *string2*, *separation*, *embeddableDescription*, *nothingInBetween=True*)
Bases: *dragonn.synthetic.synthetic.AbstractEmbeddable*

Represents a pair of strings that are embedded with some separation. Used for motif grammars. See superclass docs.

canEmbed(*priorEmbeddedThings*, *startPos*)

embedInBackgroundStringArr(*priorEmbeddedThings*, *backgroundStringArr*, *startPos*)

getDescription()

class dragonn.synthetic.synthetic.**PairEmbeddableGenerator**(*substringGenerator1*, *substringGenerator2*, *separationGenerator*, *name=None*)

Bases: *dragonn.synthetic.synthetic.AbstractEmbeddableGenerator*

generateEmbeddable()

getJsonableObject()

class dragonn.synthetic.synthetic.**PairEmbeddableGenerator_General**(*embeddableGenerator1*, *embeddableGenerator2*, *separationGenerator*, *name=None*)

Bases: *dragonn.synthetic.synthetic.AbstractEmbeddableGenerator*

generateEmbeddable()

getJsonableObject()

class dragonn.synthetic.synthetic.**PairEmbeddable_General**(*embeddable1*, *embeddable2*, *separation*, *embeddableDescription*, *nothingInBetween=True*)

Bases: *dragonn.synthetic.synthetic.AbstractEmbeddable*

embeds two Embeddable objects with some sep

canEmbed(*priorEmbeddedThings*, *startPos*)

embedInBackgroundStringArr(*priorEmbeddedThings*, *backgroundStringArr*, *startPos*)

getDescription()

class dragonn.synthetic.synthetic.**PoissonQuantityGenerator**(*mean*, *name=None*)

Bases: *dragonn.synthetic.synthetic.AbstractQuantityGenerator*

Generates values according to a poisson distribution

generateQuantity()

getJsonableObject()

class dragonn.synthetic.synthetic.**PriorEmbeddedThings_numpyArrayBacked**(*seqLen*)

Bases: *dragonn.synthetic.synthetic.AbstractPriorEmbeddedThings*

uses a numpy array where positions are set to 1 if they are occupied, to determin which positions are occupied and which are not. See parent for more documentation.

addEmbedding(*startPos*, *what*)

what: instance of Embeddable

canEmbed(*startPos*, *endPos*)

getEmbeddings()

getNumOccupiedPos()

getTotalPos()

class dragonn.synthetic.synthetic.**PwmSampler**(*pwm*, *name=None*)
Bases: *dragonn.synthetic.synthetic.AbstractSubstringGenerator*

samples from the pwm by calling self.pwm.sampleFromPwm

generateSubstring()

getJsonableObject()

class dragonn.synthetic.synthetic.**PwmSamplerFromLoadedMotifs**(*loadedMotifs*, *motifName*, *name=None*)
Bases: *dragonn.synthetic.synthetic.PwmSampler*

convenience wrapper class for instantiating parent by pulling the pwm given the name from an AbstractLoadedMotifs object (it basically extracts the pwm for you)

getJsonableObject()

class dragonn.synthetic.synthetic.**RandomSubsetOfEmbedders**(*quantityGenerator*, *embedders*, *name=None*)
Bases: *dragonn.synthetic.synthetic.AbstractEmbedder*

Takes a quantity generator that generates a quantity of embedders, and executes that many embedders from a supplied set, in sequence

getJsonableObject()

class dragonn.synthetic.synthetic.**RepeatedEmbedder**(*embedder*, *quantityGenerator*, *name=None*)
Bases: *dragonn.synthetic.synthetic.AbstractEmbedder*

Wrapper around an embedder to call it multiple times according to sampling from a distribution.

getJsonableObject()

class dragonn.synthetic.synthetic.**RepeatedSubstringBackgroundGenerator**(*substringGenerator*, *repetitions*)
Bases: *dragonn.synthetic.synthetic.AbstractBackgroundGenerator*

generateBackground()

getJsonableObject()

class dragonn.synthetic.synthetic.**ReverseComplementWrapper**(*substringGenerator*, *reverseComplementProb=0.5*, *name=None*)
Bases: *dragonn.synthetic.synthetic.AbstractSubstringGenerator*

Wrapper around a AbstractSubstringGenerator that reverse complements it with the specified probability.

generateSubstring()

getJsonableObject()

class dragonn.synthetic.synthetic.**RevertToReference**(*setOfMutations*, *name=None*)
Bases: *dragonn.synthetic.synthetic.AbstractTransformation*

for a series of mutations, reverts the supplied string to the reference ("unmutated") string

getJsonableObject()

transform(*stringArr*)

class dragonn.synthetic.synthetic.**SampleFromDiscreteDistributionSubstringGenerator**(*discreteDistrib*
Bases: *dragonn.synthetic.synthetic.AbstractSubstringGenerator*

generateSubstring()

getJsonableObject()

class dragonn.synthetic.synthetic.**StringEmbeddable**(*string*, *stringDescription=''*)
Bases: *dragonn.synthetic.synthetic.AbstractEmbeddable*

represents a string (such as a sampling from a pwm) that is to be embedded in a background. See docs for superclass.

canEmbed(*priorEmbeddedThings*, *startPos*)

embedInBackgroundStringArr(*priorEmbeddedThings*, *backgroundStringArr*, *startPos*)

classmethod fromString(*theString*)

getDescription()

class dragonn.synthetic.synthetic.**SubstringEmbeddableGenerator**(*substringGenerator*, *name=None*)
Bases: *dragonn.synthetic.synthetic.AbstractEmbeddableGenerator*

generateEmbeddable()

getJsonableObject()

class dragonn.synthetic.synthetic.**SubstringEmbedder**(*substringGenerator*, *positionGenerator=<dragonn.synthetic.synthetic.UniformPositionGenerator object>*, *name=None*)
Bases: *dragonn.synthetic.synthetic.EmbeddableEmbedder*

embeds a single generated substring within the background sequence, at a position sampled from a distribution. Only embeds at unoccupied positions

class dragonn.synthetic.synthetic.**TopNMutationsFromPwmRelativeToBestHit**(*pwm*, *N*, *bestHitMode*)
Bases: *dragonn.synthetic.synthetic.AbstractSetOfMutations*

See docs for parent; here, the collection of mutations are the top N strongest mutations for a PWM as compared to the best match for that pwm.

getJsonableObject()

class dragonn.synthetic.synthetic.**TopNMutationsFromPwmRelativeToBestHit_FromLoadedMotifs**(*loade*
pwmName
N,
bestH
Mode
Bases: *dragonn.synthetic.synthetic.TopNMutationsFromPwmRelativeToBestHit*

Like parent, except extracts the pwm.PWM object from an AbstractLoadedMotifs object, saving you a few lines of code.

getJsonableObject()

class dragonn.synthetic.synthetic.**TransformedSubstringGenerator**(*substringGenerator*, *transformations*, *transformationsDescription='transformations'*, *name=None*)

Bases: *dragonn.synthetic.synthetic.AbstractSubstringGenerator*

Takes a substringGenerator and a set of AbstractTransformation objects, applies the transformations to the generated substring

generateSubstring()

getJsonableObject()

class dragonn.synthetic.synthetic.**UniformIntegerGenerator**(*minVal*, *maxVal*, *name=None*)

Bases: *dragonn.synthetic.synthetic.AbstractQuantityGenerator*

Randomly samples an integer from minVal to maxVal, inclusive.

generateQuantity()

getJsonableObject()

class dragonn.synthetic.synthetic.**UniformPositionGenerator**(*name=None*)

Bases: *dragonn.synthetic.synthetic.AbstractPositionGenerator*

samples a start position to embed the substring in uniformly at random; does not return positions that are too close to the end of the background sequence to embed the full substring.

getJsonableObject()

class dragonn.synthetic.synthetic.**XOREmbedder**(*embedder1*, *embedder2*, *probOfFirst*, *name=None*)

Bases: *dragonn.synthetic.synthetic.AbstractEmbedder*

calls exactly one of the supplied embedders

getJsonableObject()

class dragonn.synthetic.synthetic.**ZeroInflater**(*quantityGenerator*, *zeroProb*, *name=None*)

Bases: *dragonn.synthetic.synthetic.AbstractQuantityGenerator*

Wrapper that inflates the number of zeros returned. Flips a coin; if positive, will return zero - otherwise will sample from the wrapped distribution (which may still return 0)

generateQuantity()

getJsonableObject()

class dragonn.synthetic.synthetic.**ZeroOrderBackgroundGenerator**(*seqLength*, *discreteDistribution=<dragonn.synthetic.util.DiscreteDistri object>*)

Bases: *dragonn.synthetic.synthetic.RepeatedSubstringBackgroundGenerator*

returns a sequence with 40% GC content. Each base is sampled independently.

dragonn.synthetic.synthetic.**generateString**(*options*)

dragonn.synthetic.synthetic.**generateString_zeroOrderMarkov**(*length*, *discreteDistribution=<dragonn.synthetic.util.DiscreteDistribut object>*)

discreteDistribution: instance of util.DiscreteDistribution

dragonn.synthetic.synthetic.**getEmbeddingsFromString**(*string*)

dragonn.synthetic.synthetic.**getFileNamePieceFromOptions**(*options*)

dragonn.synthetic.synthetic.**getGenerationOption**(*string*)

dragonn.synthetic.synthetic.**getParentArgparse**()

dragonn.synthetic.synthetic.**printSequences**(*outputFileName*, *sequenceSetGenerator*, *includeEmbeddings=False*, *labelGenerator=None*, *includeFasta=False*)

outputFileName: string sequenceSetGenerator: instance of AbstractSequenceSetGenerator Given an output filename, and an instance of AbstractSequenceSetGenerator, will call the sequence set generator and print the generated sequences to the output file. Will also create a file "info_outputFileName.txt" in the samedirectory as outputFileName that contains all the information about sequenceSetGenerator. includeEmbeddings: a boolean indicating whether to print a column that lists the embeddings labelGenerator: instance of LabelGenerator

dragonn.synthetic.synthetic.**printSequencesTransformationPosNeg**(*outputFileNamePos*, *outputFileNameNeg*, *sequenceSetGenerator*, *transformation*)

outputFileName: string sequenceSetGenerator: instance of AbstractSequenceSetGenerator

generatedSequences: the sequences that have been generated by sequenceSetGenerator

Given an output filename, and an instance of AbstractSequenceSetGenerator, will print the generated sequences to the output file. Will also create a file

"info_outputFileName.txt" in the same directory as outputFileName that contains all the information about sequenceSetGenerator.

dragonn.synthetic.synthetic.**sampleIndexWithinRegionOfLength**(*length*, *lengthOfThingToEmbed*)

uniformly at random samples integers from 0 to length-lengthOfThingToEmbedIn

dragonn.synthetic.util module

class dragonn.synthetic.util.**ArgParseArgument**(*argumentName*, ***kwargs*)

Bases: object

addToParser(*parser*)

class dragonn.synthetic.util.**ArgsAndKwargs**(*args*, *kwargs*)

Bases: tuple

args

Alias for field number 0

kwargs

Alias for field number 1

class dragonn.synthetic.util.**ArgumentToAdd**(*val*, *argumentName=None*, *argNameAndValSep='-'*)

Bases: object

Class to append runtime arguments to a string to facilitate auto-generation of output file names.

argNamePrefix()

transform()

class dragonn.synthetic.util.**ArrArgument**(*val*, *argumentName*, *sep='+'*, *toStringFunc=<type 'str'>*)
Bases: *dragonn.synthetic.util.ArgumentToAdd*

transform()

class dragonn.synthetic.util.**ArrOfFileNamesArgument**(*val*, *argumentName*, *sep='+'*)
Bases: *dragonn.synthetic.util.ArrArgument*

class dragonn.synthetic.util.**BooleanArgument**(*val*, *argumentName=None*, *argNameAndValSep='-'*)
Bases: *dragonn.synthetic.util.ArgumentToAdd*

transform()

dragonn.synthetic.util.**CROSSC_NORMFUNC**
alias of Enum

class dragonn.synthetic.util.**CoreFileNameArgument**(*val*, *argumentName=None*, *argNameAndValSep='-'*)
Bases: *dragonn.synthetic.util.ArgumentToAdd*

transform()

class dragonn.synthetic.util.**DiscreteDistribution**(*valToFreq*)
Bases: object

class dragonn.synthetic.util.**Entity**(*id*)
Bases: object

addAttribute(*attributeName*, *value*)

getAttribute(*attributeName*)

hasAttribute(*attributeName*)

class dragonn.synthetic.util.**GetBest**
Bases: object

getBest()

getBestObj()

getBestVal()

isBetter(*val*)

process(*theObject*, *val*)

class dragonn.synthetic.util.**GetBest_Max**
Bases: *dragonn.synthetic.util.GetBest*

isBetter(*val*)

class dragonn.synthetic.util.**GetBest_Min**
Bases: *dragonn.synthetic.util.GetBest*

isBetter(*val*)

class dragonn.synthetic.util.**IterableFromDict**(*theDict*, *defaultVal*, *totalLen*)
Bases: object

next()

class dragonn.synthetic.util.**Options**(***kwargs*)
Bases: object

dragonn.synthetic.util.**PERPOS_NORMFUNC**
alias of Enum

class dragonn.synthetic.util.**SparseArrFromDict**(*theDict*, *defaultVal*, *totalLen*)
Bases: object

dragonn.synthetic.util.**SplitNames**
alias of Enum

class dragonn.synthetic.util.**TeeOutputStreams**(**streams*)
Bases: object

for piping to several output streams

close()

closed

flush()

write(*data*)

writeable()

writelines(*lines*)

class dragonn.synthetic.util.**Titled2DMatrix**(*colNamesPresent=False*, *rowNamesPresent=False*, *rows=None*, *colNames=None*, *rowNames=None*)
Bases: object

has a 2D matrix, rowNames and colNames arrays

addRow(*arr*, *rowName=None*)

normaliseRows()

printToFile(*fileHandle*)

setColNames(*colNames*)

class dragonn.synthetic.util.**TitledArr**(*title*, *arr*, *colNameToIndex=None*)
Bases: object

getCol(*colName*)

setCol(*colName*, *value*)

class dragonn.synthetic.util.**TitledMapping**(*titleArr*, *flagIfInconsistent=False*)
Bases: object

When each key maps to an array, and each index in the array is associated with a name.

addKey(*key*, *arr*)

getArrForKey(*key*)

getTitledArrForKey(*key*)
returns an instance of util.TitledArr which has: getCol(colName) and setCol(colName)

keyPresenceCheck(*key*)
Throws an error if the key is absent

printToFile(*fileHandle*, *includeRownames=True*)

class dragonn.synthetic.util.**TitledMappingIterator**(*titledMapping*)
Bases: object

Returns an iterator over TitledArrs for the keys in titledMapping.mapping

next()

class dragonn.synthetic.util.**VariableWrapper**(*var*)
For when I want reference-type access to an immutable

dragonn.synthetic.util.**absentOrNone**(*obj*, *attr*)

dragonn.synthetic.util.**addArguments**(*string*, *args*, *joiner='_'*)
args is an array of ArgumentToAdd.

dragonn.synthetic.util.**addDictionary**(*toUpdate*, *toAdd*, *initVal=0*, *mergeFunc=<function <lambda>>*)
Defaults to addition, technically applicable any time you want to update a dictionary (toUpdate) with the entries of another dictionary (toAdd) using a particular operation (eg: adding corresponding keys)

dragonn.synthetic.util.**arrToDict**(*arr*)
Turn an array into a dictionary where each value maps to '1' used for membership testing.

dragonn.synthetic.util.**arrayEquals**(*arr1*, *arr2*)
compares corresponding entries in arr1 and arr2

dragonn.synthetic.util.**assertAllOrNone**(*obj*, *attrNames*)

dragonn.synthetic.util.**assertArrayElementsEqual**(*arr1*, *arr2*, *threshold=0.0*)

dragonn.synthetic.util.**assertAtLeastOneSet**(*obj*, *attrs*)

dragonn.synthetic.util.**assertAttributesHaveTheSameLengths**(*attributes*, *attributeNames*)

dragonn.synthetic.util.**assertDoesNotHaveAttributes**(*obj*, *attributes*, *explanation*)

dragonn.synthetic.util.**assertHasAttributes**(*obj*, *attributes*, *explanation*)

dragonn.synthetic.util.**assertIsType**(*instance*, *theClass*, *instanceVarName*)

dragonn.synthetic.util.**assertLessThanOrEqual**(*obj*, *smallerAttrName*, *largerAttrName*)

dragonn.synthetic.util.**assertMutuallyExclusiveAttributes**(*obj*, *attrs*)

dragonn.synthetic.util.**auPRC**(*trueY*, *predictedYscores*, *plotFileName=None*)

dragonn.synthetic.util.**augmentArgparseKwargsHelpWithDefault**(***argParseKwargs*)

dragonn.synthetic.util.**autovivisect**(*theDict*, *getThingToInitialiseWith*, **keys*)

dragonn.synthetic.util.**avgNumpyArrays**(*numpyArrays*)

dragonn.synthetic.util.**chainFunctions**(**functions*)

dragonn.synthetic.util.**checkForAttributes**(*item*, *attributesToCheckFor*, *itemName=None*)

dragonn.synthetic.util.**check_pid**(*pid*)
Check For the existence of a unix pid.

dragonn.synthetic.util.**combineEnums**(**enums*)

dragonn.synthetic.util.**computeConfusionMatrix**(*actual*, *predictions*, *labelOrdering=None*)

dragonn.synthetic.util.**computeCooccurence**(*matrix*)
matrix: rows (first dim) are examples

dragonn.synthetic.util.**computeRunningWindowMax**(*arr*, *windowSize*)

dragonn.synthetic.util.**computeRunningWindowMaxActivation_2d**(*arr*, *smallerArr*, *windowSize*)

dragonn.synthetic.util.**computeRunningWindowOneOverMaxActivation_2d**(*arr*, *smallerArr*, *windowSize*)

dragonn.synthetic.util.**computeRunningWindowOneOverTwoNorm_2d**(*arr*, *smallerArr*, *windowSize*)

dragonn.synthetic.util.**computeRunningWindowOp**(*arr*, *windowSize*, *op*)

dragonn.synthetic.util.**computeRunningWindowSum**(*arr*, *windowSize*)

dragonn.synthetic.util.**computeRunningWindowSum_2d**(*arr*, *smallerArr*, *windowSize*)

dragonn.synthetic.util.**computeRunningWindowTwoNorm_2d**(*arr*, *smallerArr*, *windowSize*)

dragonn.synthetic.util.**crossCorrelateArraysLengthwise**(*arr1*, *arr2*, *normaliseFunc*, *smallerPerPosNormFuncs=[]*, *largerPerPosNormFuncs=[]*, *auxLargerForPerPosNorm=None*, *auxLargerPerPosNormFuncs=[]*, *pad=True*)

dragonn.synthetic.util.**crossCorrelation_2d**(*arr*, *smallerArr*, *windowSize*)

dragonn.synthetic.util.**defaultTransformation**()

dragonn.synthetic.util.**dict2str**(*theDict*, *sep='\n'*)

dragonn.synthetic.util.**divideByPerPositionRange**(*arr*)

dragonn.synthetic.util.**doPCAonFile**(*theFile*)

dragonn.synthetic.util.**doesNotWorkForMultithreading_redirectStdout**(*func*, *redirectedStdout*)

dragonn.synthetic.util.**enum**(***enums*)

dragonn.synthetic.util.**enumerate_skipFirst**(*aList*)

dragonn.synthetic.util.**executeAsSystemCall**(*commandToExecute*)

dragonn.synthetic.util.**executeForAllFilesInDirectory**(*directory*, *function*, *fileFilterFunction=<function <lambda>>*)

dragonn.synthetic.util.**floatRange**(*start*, *end*, *step*)

Like range but for floats...

dragonn.synthetic.util.**formatDictAsArgsString**(*theDict*, *subDictEnclosingChars=""*)

dragonn.synthetic.util.**formattedJsonDump**(*jsonData*)

dragonn.synthetic.util.**fracToRainbowColour**(*frac*)

frac is a number from 0 to 1. Map to a 3-tuple representing a rainbow colour.

1 -> (0, 1, 0) #green 0.75 -> (1, 0, 1) #yellow 0.5 -> (1, 0, 0) #red 0.25 -> (1, 1, 0) #violet 0 -> (0, 0, 1) #blue

dragonn.synthetic.util.**getAllPossibleSubsets**(*arr*)

dragonn.synthetic.util.**getBest**(*arr*, *getterFunc*, *takeMax*)

Will return a tuple of the index and the value of the best as extracted by getterFunc

dragonn.synthetic.util.**getBestLengthwiseCrossCorrelationOfArrays**(*arr1*, *arr2*, *normaliseFunc*, *smallerPerPosNormFuncs*, *largerPerPosNormFuncs*)

dragonn.synthetic.util.**getDateTimeString**(*datetimeFormat='%y-%m-%d-%H-%M'*)

dragonn.synthetic.util.**getErrorTraceback**()

dragonn.synthetic.util.**getExtremeN**(*toSort*, *N*, *keyFunc*)

Returns the indices

dragonn.synthetic.util.**getFromEnum**(*theEnum*, *enumName*, *string*)

dragonn.synthetic.util.**getIntervals**(*minVal*, *numSteps*, ***kwargs*)

dragonn.synthetic.util.**getMaxIndex**(*arr*)

dragonn.synthetic.util.**getNthInterval**(*minVal*, *maxVal*, *numSteps*, *n*, *logarithmic*, *roundTo*, *cast*)

logarithmic: boolean indicating if want log numSteps vs linear roundTo: can be set to None for no rounding cast: can be set to just lambda x: x

dragonn.synthetic.util.**getRandomString**(*size*)

dragonn.synthetic.util.**getSingleton**(*name*)

dragonn.synthetic.util.**getStackTrace**()

dragonn.synthetic.util.**getTempDir**()

dragonn.synthetic.util.**imageToSeq**(*image*)

dragonn.synthetic.util.**intersects**(*chromStartEnd1*, *chromStartEnd2*)

dragonn.synthetic.util.**invertIndices**(*selectedIndices*, *fullSetOfIndices*)

Returns all indices in fullSet but not in selected.

dragonn.synthetic.util.**invertPermutation**(*permutation*)

dragonn.synthetic.util.**isBetter**(*value*, *referenceValue*, *isLargerBetter*)

dragonn.synthetic.util.**isBetterOrEqual**(*value*, *referenceValue*, *isLargerBetter*)

dragonn.synthetic.util.**isNumpy**(*obj*)

dragonn.synthetic.util.**iter_skipFirst**(*aList*)

dragonn.synthetic.util.**linecount**(*filename*)

dragonn.synthetic.util.**makeChromStartEnd**(*chrom*, *start*, *end*)

dragonn.synthetic.util.**multiprocessing_map_printProgress**(*secondsBetweenUpdates*, *numThreads*, *func*, *iterable*)

dragonn.synthetic.util.**normaliseByRowsAndColumns**(*theMatrix*)

The matrix is as a dictionary

dragonn.synthetic.util.**normaliseEntriesByMeanAndSdev**(*arr*)

dragonn.synthetic.util.**normaliseEntriesByMeanAndTwoNorm**(*arr*)

dragonn.synthetic.util.**normaliseEntriesBySdev**(*arr*)

dragonn.synthetic.util.**normaliseEntriesByTwoNorm**(*arr*)

dragonn.synthetic.util.**normaliseEntriesZeroToOne**(*arr*)

dragonn.synthetic.util.**normaliseRowsByMeanAndSdev**(*arr*)

dragonn.synthetic.util.**normaliseRowsByMeanAndSdev_firstFourSeq**(*arr*)

dragonn.synthetic.util.**npArrayIfList**(*arr*)

dragonn.synthetic.util.**objectFromArgsAndKwargs**(*classOfObject*, *args=[]*, *kwargs={}*)

dragonn.synthetic.util.**objectFromArgsAndKwargsFromYaml**(*classOfObject*, *yamlWithArgsAndKwargs*)

dragonn.synthetic.util.**overrides**(*interface_class*)

dragonn.synthetic.util.**parseJsonFile**(*fileName*)

dragonn.synthetic.util.**parseMultipleYamlFiles**(*pathsToYaml*)

dragonn.synthetic.util.**parseYamlFile**(*fileName*)

dragonn.synthetic.util.**parseYamlFileHandle**(*fileHandle*)

dragonn.synthetic.util.**plotPRC**(*precision*, *recall*, *auc*, *plotFileName*)

dragonn.synthetic.util.**presentAndNotNone**(*obj*, *attr*)

dragonn.synthetic.util.**presentAndNotNoneOrFalse**(*obj*, *attr*)

dragonn.synthetic.util.**printAttributes**(*entities*, *attributesToPrint*, *outputFile*)

dragonn.synthetic.util.**printCoordinatesForLabelSubsets**(*regionIds*, *labels*, *labelSetsToFilterFor*, *outputFilePrefix*)

assumes regionIds of the form chr:start-end labelSetsToFilter as an iterable of iterables of the

label you want to subset. Will be incorportated into the filename

outputFile will be outputFilePrefix+"_" +"-".join(str(x) for x in labelsToFilter)

dragonn.synthetic.util.**printRegionIds**(*regionIds*, *labels*, *labelFilter*, *outputFile*, *idTransformation=<function <lambda>>*)

dragonn.synthetic.util.**randomlySampleFromArr**(*arr*)

dragonn.synthetic.util.**readMultilineRawInput**(*prompt*)

dragonn.synthetic.util.**redirectStdoutToString**(*func*, *logger=None*, *emailErrorFunc=None*)

dragonn.synthetic.util.**reverseComplement**(*sequence*)

dragonn.synthetic.util.**reverse_enumerate**(*aList*)

dragonn.synthetic.util.**roundToNearest**(*val*, *nearest*)

dragonn.synthetic.util.**rowNormalise**(*matrix*)

dragonn.synthetic.util.**runningWindowOneOver**(*func*, *arr*, *smallerArr*, *windowSize*)

dragonn.synthetic.util.**sampleFromDiscreteDistribution**(*discereteDistribution*)

Expecting an instance of DiscreteDistribution

dragonn.synthetic.util.**sampleFromNumSteps**(*numSteps*, ***kwargs*)

dragonn.synthetic.util.**sampleFromProbsArr**(*arrWithProbs*)
Will return a sampled index

dragonn.synthetic.util.**sampleFromRangeWithStepSize**(*minVal*, *maxVal*, *stepSize*, *cast*)
cast can be just max-min

dragonn.synthetic.util.**sampleNinstancesFromDiscreteDistribution**(*N*, *discreteDistribution*)

dragonn.synthetic.util.**sampleWithoutReplacement**(*arr*, *numToSample*)

dragonn.synthetic.util.**sendEmail**(*to*, *frm*, *subject*, *contents*)

dragonn.synthetic.util.**sendEmails**(*tos*, *frm*, *subject*, *contents*)

dragonn.synthetic.util.**seqTo2DImages_fillInArray**(*zerosArray*, *sequence*)

dragonn.synthetic.util.**seqTo2Dimage**(*sequence*)

dragonn.synthetic.util.**setOfSeqsTo2Dimages**(*sequences*)

dragonn.synthetic.util.**shuffleArray**(**arrs*)

dragonn.synthetic.util.**sortByLabels**(*arr*, *labels*)

intended use case: sorting by cluster labels for plotting a heatmap

dragonn.synthetic.util.**splitChromStartEnd**(*chromId*)

dragonn.synthetic.util.**splitIgnoringQuotes**(*string*, *charToSplitOn=' '*)
will split on charToSplitOn, ignoring things that are in quotes

dragonn.synthetic.util.**splitIntegerIntoProportions**(*integer*, *proportions*)

dragonn.synthetic.util.**submitProcess**(*command*)

dragonn.synthetic.util.**sumAbsDifferences**(*arr1*, *arr2*)

dragonn.synthetic.util.**sumNumpyArrays**(*numpyArrays*)

dragonn.synthetic.util.**swapIndices**(*arr*, *idx1*, *idx2*)

dragonn.synthetic.util.**throwErrorIfUnequalSets**(*given*, *expected*)

dragonn.synthetic.util.**transformType**(*inp*, *theType*)

dragonn.synthetic.util.**valToIndexMap**(*arr*)
A map from a value to the index at which it occurs

dragonn.synthetic.util.**yamlToArgsString**(*yamlString*, *subDictEnclosingChars=""*)

Module contents

1.2 Submodules

1.2.1 dragonn.hyperparameter_search module

1.2.2 dragonn.metrics module

class dragonn.metrics.**ClassificationResult**(*labels*, *predictions*, *task_names=None*)
Bases: object

class dragonn.metrics.**IgnoreNumpyErrors**
Bases: object

dragonn.metrics.**auPRC**(*labels*, *predictions*)

dragonn.metrics.**auPRG**(*labels*, *predictions*)

dragonn.metrics.**auROC**(*labels*, *predictions*)

dragonn.metrics.**balanced_accuracy**(*labels*, *predictions*, *threshold=0.5*)

dragonn.metrics.**negative_accuracy**(*labels*, *predictions*, *threshold=0.5*)

dragonn.metrics.**positive_accuracy**(*labels*, *predictions*, *threshold=0.5*)

dragonn.metrics.**recall_at_precision_threshold**(*labels*, *predictions*, *precision_threshold*)

1.2.3 dragonn.models module

class dragonn.models.**DecisionTree**
Bases: *dragonn.models.Model*

predict(*X*)

train(*X*, *y*, *validation_data=None*)

class dragonn.models.**Model**(***hyperparameters*)
Bases: object

predict(*X*)

score(*X*, *y*, *metric*)

test(*X*, *y*)

train(*X*, *y*, *validation_data*)

class dragonn.models.**MotifScoreRNN**(*input_shape*, *gru_size=10*, *tdd_size=4*)
Bases: *dragonn.models.Model*

predict(*X*)

train(*X*, *y*, *validation_data*)

class dragonn.models.**RandomForest**
Bases: *dragonn.models.DecisionTree*

class dragonn.models.**SVC**
Bases: *dragonn.models.Model*

predict(*X*)

train(*X*, *y*, *validation_data=None*)

class dragonn.models.**SequenceDNN**(*seq_length*, *use_deep_CNN=False*, *use_RNN=False*, *num_tasks=1*, *num_filters=15*, *conv_width=15*, *num_filters_2=15*, *conv_width_2=15*, *num_filters_3=15*, *conv_width_3=15*, *pool_width=35*, *L1=0*, *dropout=0.0*, *GRU_size=35*, *TDD_size=15*, *verbose=1*)
Bases: *dragonn.models.Model*

Sequence DNN models.

seq_length [int] length of input sequence.

use_deep_CNN [bool, optional] uses 3 layered CNN if True, 1 layered CNN if False. Default: False.

num_tasks [int,] number of tasks. Default: 1.

num_filters [int] number of 1st layer convolutional filters. Default: 15.

conv_width [int] width of 1st layer convolutional filters. Default: 15.

pool_width [int] width of max pooling. Default: 35.

num_filters_2 [int] number of 2nd layer convolutional filters. Default: 15.

conv_width_2 [int] width of 2nd layer convolutional filters. Default: 15.

num_filters_3 [int] number of 3rd layer convolutional filters. Default: 15.

conv_width_3 [int] width of 3rd layer convolutional filters. Default: 15.

L1 [float] strength of L1 penalty.

dropout [float] dropout probability in every convolutional layer. Default: 0.

num_tasks [int] Number of prediction tasks or labels. Default: 1.

verbose: int Verbosity level during training. Valida values: 0, 1, 2.

Compiled DNN model.

class `LossHistory`(*X_train*, *y_train*, *validation_data*, *sequence_DNN*)
Bases: `keras.callbacks.Callback`

`on_epoch_end`(*epoch*, *logs={}*)

class `SequenceDNN.PrintMetrics`(*validation_data*, *sequence_DNN*)
Bases: `keras.callbacks.Callback`

`on_epoch_end`(*epoch*, *logs={}*)

`SequenceDNN.deeplift`(*X*, *batch_size=200*)
Returns (num_task, num_samples, input_shape) deeplift score array.

`SequenceDNN.get_sequence_filters`()
Returns list with sequence filter 2darrays.

`SequenceDNN.in_silico_mutagenesis`(*X*)

`SequenceDNN.predict`(*X*)

`SequenceDNN.train`(*X*, *y*, *validation_data*)

class `dragonn.models.gkmSVM`(*prefix='./gkmSVM'*, *word_length=11*, *mismatches=3*, *C=1*)
Bases: *dragonn.models.Model*

static `encode_sequence_into_fasta_file`(*sequence_iterator*, *ofname*)
writes sequences into fasta file

`model_file`

`predict`(*X*)

`train`(*X*, *y*, *validation_data=None*)
Trains gkm-svm, saves model file.

1.2.4 dragonn.plot module

class `dragonn.plot.Polygon`(*context*)
Bases: `object`

`exterior`

`geom_type`

interiors

dragonn.plot.**PolygonPatch**(*polygon*, ***kwargs*)
Constructs a matplotlib patch from a geometric object

The *polygon* may be a Shapely or GeoJSON-like object with or without holes. The *kwargs* are those supported by the matplotlib.patches.Polygon class constructor. Returns an instance of matplotlib.patches.PathPatch.

Example (using Shapely Point and a matplotlib axes):

```
>>> b = Point(0, 0).buffer(1.0)
>>> patch = PolygonPatch(b, fc='blue', ec='blue', alpha=0.5)
>>> axis.add_patch(patch)
```

dragonn.plot.**PolygonPath**(*polygon*)
Constructs a compound matplotlib path from a Shapely or GeoJSON-like geometric object

dragonn.plot.**add_letter_to_axis**(*ax*, *let*, *x*, *y*, *height*)
Add 'let' with position x,y and height height to matplotlib axis 'ax'.

dragonn.plot.**add_letters_to_axis**(*ax*, *letter_heights*)
Plots letter on user-specified axis.

ax : axis letter_heights: Nx4 array

dragonn.plot.**plot_bases**(*letter_heights*, *figsize=(12, 6)*, *ylab='bits'*)
Plot the N letters with heights taken from the Nx4 matrix letter_heights.

letter_heights: Nx4 array ylab: y axis label

pyplot figure

dragonn.plot.**plot_motif**(*motif_name*, *figsize*, *ylab='bits'*, *information_content=True*)
Plot motifs from encode motifs file

dragonn.plot.**plot_pwm**(*letter_heights*, *figsize=(12, 6)*, *ylab='bits'*, *information_content=True*)
Plots pwm. Displays information content by default.

dragonn.plot.**standardize_polygons_str**(*data_str*)
Given a POLYGON string, standardize the coordinates to a 1x1 grid.

Input : data_str (taken from above) Output: tuple of polygon objects

1.2.5 dragonn.simulations module

dragonn.simulations.**get_distribution**(*GC_fraction*)

dragonn.simulations.**motif_density**(*motif_name*, *seq_length*, *num_seqs*, *min_counts*, *max_counts*, *GC_fraction*, *central_bp=None*)
returns sequences with motif density.

dragonn.simulations.**simple_motif_embedding**(*motif_name*, *seq_length*, *num_seqs*, *GC_fraction*)
returns sequence array

dragonn.simulations.**simulate_differential_accessibility**(*pos_motif_names*, *neg_motif_names*, *seq_length*, *min_num_motifs*, *max_num_motifs*, *num_pos*, *num_neg*, *GC_fraction*)
Generates data for differential accessibility task.

pos_motif_names [list] List of strings.

neg_motif_names [list] List of strings.

seq_length : int min_num_motifs : int max_num_motifs : int num_pos : int num_neg : int GC_fraction : float

sequence_arr [1darray] Contains sequence strings.

y [1darray] Contains labels.

dragonn.simulations.**simulate_heterodimer_grammar**(*motif1*, *motif2*, *seq_length*, *min_spacing*, *max_spacing*, *num_pos*, *num_neg*, *GC_fraction*)

Simulates two classes of sequences with motif1 and motif2:

- Positive class sequences with motif1 and motif2 positioned min_spacing and max_spacing
- Negative class sequences with independent motif1 and motif2 positioned

anywhere in the sequence, not as a heterodimer grammar

seq_length : int, length of sequence GC_fraction : float, GC fraction in background sequence num_pos : int, number of positive class sequences num_neg : int, number of negatice class sequences motif1 : str, encode motif name motif2 : str, encode motif name min_spacing : int, minimum inter motif spacing max_spacing : int, maximum inter motif spacing

sequence_arr [1darray] Array with sequence strings.

y [1darray] Array with positive/negative class labels.

dragonn.simulations.**simulate_motif_counting**(*motif_name*, *seq_length*, *pos_counts*, *neg_counts*, *num_pos*, *num_neg*, *GC_fraction*)

Generates data for motif counting task. Parameters ———- motif_name : str seq_length : int pos_counts : list

(min_counts, max_counts) for positive set.

neg_counts [list] (min_counts, max_counts) for negative set.

num_pos : int num_neg : int GC_fraction : float Returns ——- sequence_arr : 1darray

Contains sequence strings.

y [1darray] Contains labels.

dragonn.simulations.**simulate_motif_density_localization**(*motif_name*, *seq_length*, *center_size*, *min_motif_counts*, *max_motif_counts*, *num_pos*, *num_neg*, *GC_fraction*)

Simulates two classes of seqeuences:

- Positive class sequences with multiple motif instances in center of the sequence.
- Negative class sequences with multiple motif instances anywhere in the sequence.

The number of motif instances is uniformly sampled between minimum and maximum motif counts.

motif_name [str] encode motif name

seq_length [int] length of sequence

center_size [int] length of central part of the sequence where motifs can be positioned

min_motif_counts [int] minimum number of motif instances

max_motif_counts [int] maximum number of motif instances

num_pos [int] number of positive class sequences

num_neg [int] number of negative class sequences

GC_fraction [float] GC fraction in background sequence

sequence_arr [1darray] Contains sequence strings.

y [1darray] Contains labels.

dragonn.simulations.**simulate_multi_motif_embedding**(*motif_names*, *seq_length*, *min_num_motifs*, *max_num_motifs*, *num_seqs*, *GC_fraction*)

Generates data for multi motif recognition task. Parameters ———- motif_names : list

List of strings.

seq_length : int min_num_motifs : int max_num_motifs : int num_seqs : int GC_fraction : float Returns ——- sequence_arr : 1darray

Contains sequence strings.

y [ndarray] Contains labels for each motif.

dragonn.simulations.**simulate_single_motif_detection**(*motif_name*, *seq_length*, *num_pos*, *num_neg*, *GC_fraction*)

Simulates two classes of seqeuences:

- Positive class sequence with a motif embedded anywhere in the sequence
- Negative class sequence without the motif

motif_name [str] encode motif name

seq_length [int] length of sequence

num_pos [int] number of positive class sequences

num_neg [int] number of negative class sequences

GC_fraction [float] GC fraction in background sequence

sequence_arr [1darray] Array with sequence strings.

y [1darray] Array with positive/negative class labels.

1.2.6 dragonn.tutorial_utils module

class dragonn.tutorial_utils.**Data**(*X_train*, *X_valid*, *X_test*, *y_train*, *y_valid*, *y_test*, *motif_names*)

Bases: tuple

X_test

Alias for field number 2

X_train
Alias for field number 0

X_valid
Alias for field number 1

motif_names
Alias for field number 6

y_test
Alias for field number 5

y_train
Alias for field number 3

y_valid
Alias for field number 4

dragonn.tutorial_utils.**SequenceDNN_learning_curve**(*dnn*)

dragonn.tutorial_utils.**get_SequenceDNN**(*SequenceDNN_parameters*)

dragonn.tutorial_utils.**get_available_simulations**()

dragonn.tutorial_utils.**get_simulation_data**(*simulation_name*, *simulation_parameters*, *test_set_size=4000*, *validation_set_size=3200*)

dragonn.tutorial_utils.**get_simulation_function**(*simulation_name*)

dragonn.tutorial_utils.**inspect_SequenceDNN**()

dragonn.tutorial_utils.**interpret_SequenceDNN_distributed**(*dnn*, *simulation_data*, *plot_layer_outputs=False*)

dragonn.tutorial_utils.**interpret_SequenceDNN_integrative**(*dnn*, *simulation_data*)

dragonn.tutorial_utils.**plot_SequenceDNN_layer_outputs**(*dnn*, *simulation_data*)

dragonn.tutorial_utils.**plot_motifs**(*simulation_data*)

dragonn.tutorial_utils.**plot_sequence_filters**(*dnn*)

dragonn.tutorial_utils.**print_available_simulations**()

dragonn.tutorial_utils.**print_simulation_info**(*simulation_name*)

dragonn.tutorial_utils.**test_SequenceDNN**(*dnn*, *simulation_data*)

dragonn.tutorial_utils.**train_SequenceDNN**(*dnn*, *simulation_data*)

1.2.7 dragonn.utils module

dragonn.utils.**get_motif_scores**(*encoded_sequences*, *motif_names*, *max_scores=None*, *return_positions=False*, *GC_fraction=0.4*)

Computes pwm log odds.

encoded_sequences : 4darray motif_names : list of strings max_scores : int, optional return_positions : boolean, optional GC_fraction : float, optional

(num_samples, seq_length, num_motifs) complete score array by default. If max_scores, (num_samples, num_motifs*max_scores) max score array. If max_scores and return_positions, (num_samples, 2*num_motifs*max_scores) array with max scores and their positions.

dragonn.utils.**get_pssm_scores**(*encoded_sequences*, *pssm*)

dragonn.utils.**one_hot_encode**(*sequences*)

dragonn.utils.**reverse_complement**(*encoded_seqs*)

1.3 Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

d

A

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z