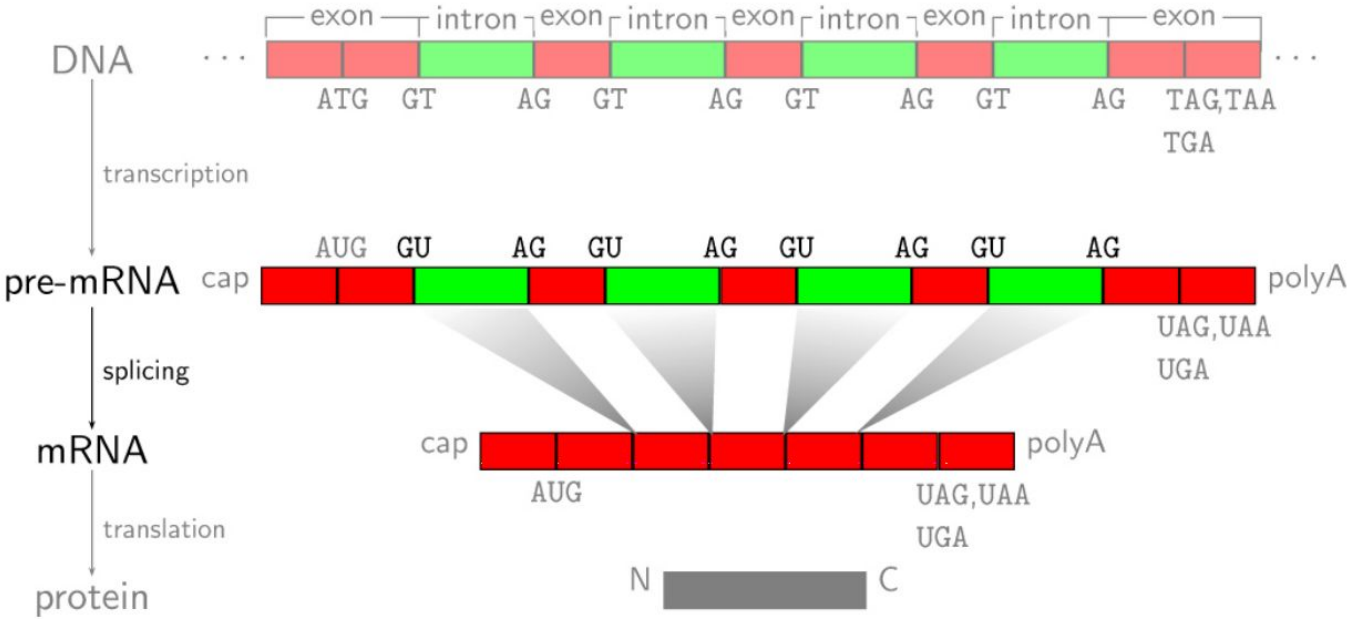
Exercise session - Project 4
ML4H - 30.04.2020

Splice Sites



- Almost all *donor splice sites* exhibit `GU`
- Almost all *acceptor splice site* exhibit `AG`
- Not all `GU`s and `AG`s are used as splice site

The project

- Two datasets:
  - C. Elegans DNA dataset.
  - Human DNA dataset.
- Both datasets contain:
  - DNA sequences.
  - Splice site annotations.
- Find the best model possible to predict splice site given the sequence.
- Groups of up to 3

C. Elegans DNA

- You get a csv file, with two columns: sequences and labels.
- 2200 samples.
- Sequences of 82 nucleotides.
- 200 true labels, 9% of the samples.
- Do your own experimental setup.
- Inside a jupyter notebook:
  - Do your own experimental setup (cross validation, random split, etc.).
  - Find best model possible (try different models, do hyperparameter search, etc).
  - ROC and PRC curves of your best model.
  - Pandas dataframe showing a comparison of the scores of the models you tried. Three columns: model name, AUROC, AUPRC, scores for the test set.

Human DNA

- You get 4 CSV files: train, validation, test and hidden test splits.
- Train CSV contains 500K samples, validation CSV contains 30K samples and test CSV contains 30K samples (90%, 5%, 5%). Hide test CSV file also contains 30K samples.
- Sequences of 398 nucleotides.
- Use the validation set for whatever you want.
- Don’t use the test set for training, or hyperparameter search, etc!
- Inside Jupyter:
  - Find best model possible (try different models, do hyperparameter search, etc).
  - ROC and PRC curves of your best model.
  - Pandas dataframe showing a comparison of the scores of the models you tried. Three columns: model name, AUROC, AUPRC, scores for the test set.
  - Predictions for the hide test dataset saved as a 1D numpy array with np.save and “result.npy” as filename.

Jupyter notebook general advice

- Sequential execution without gaps.
- Don’t use the test set for training or model selection!
- Do a nice organization of your code and notebook (use markdown titles, explanatory texts, etc.).

Deliverables

- Jupyter notebook for C. Elegans DNA dataset.
- Jupyter notebook for Human DNA dataset.
- results.npy predictions for hidden human DNA dataset.
- Conda yaml environment.
- Max 2 page report, including methods, results and conclusions plus the names of the group members and their main contribution.

Evaluation

- **Scores on the hidden test set.**
- Scores on the test sets.
- Quality of the experiments.
- Number of models you tried (try also simple models!).
- Project organization and quality of the documentation.

Deadline

20/05/2020

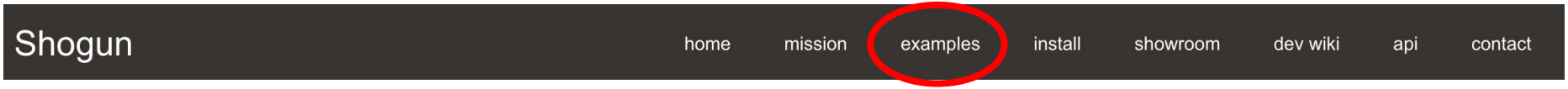
	sequences	labels
0	TTGTGTCCTACTTTTGTCCATTTGGAAAAATAATTGCATGACTACA...	-1
1	CTTTCCTTTATTTCTTCGTCAACTTAATATCCTTAGCAAAACAGGA...	-1
2	TACTTAAGAGGGGTAAGAAATATATAAACTAGTGCAACATTTTTCA...	-1
3	TAGGTTTCCAAGCAGCCCATTCCTGCCTGGCACCACAGGGATCCAT...	-1
4	GCATGAGCCACTGCGCCTGGCCTGGTTCATTGCTTCTTAGTGATGC...	-1

- Only potential acceptor sites, i.e. a string with AG in the middle of the string.

Shogun

http://shogun-toolbox.org/

```
conda install -c conda-forge shogun
```



Kernel Support Vector Machine

w

CWaveKernel (shogun)
CWaveletKernel (shogun)
CWDFeatures (shogun)
CWeightedCommWordStringKernel (shogun)
CWeightedDegreePositionStringKernel (shogun)
CWeightedDegreeRBFKernel (shogun)
CWeightedDegreeStringKernel (shogun)
CWeightedMajorityVote (shogun)
CWRACCMeasure (shogun)
Wrapper

CWeightedDegreeStringKernel Class Reference

List of all members | Public Types | Public Member Functions | Static Public Member Functions | Public Attributes | Protected Member Functions | Static Protected Member Functions | Protected Attributes

Detailed Description

The Weighted Degree String kernel.

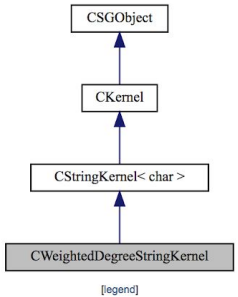
The WD kernel of order d compares two sequences $\mathbf{x}$ and $\mathbf{x}'$ of length L by summing all contributions of k-mer matches of lengths $k \in \{1, \ldots, d\}$, weighted by coefficients β_k. It is defined as

$$k(\mathbf{x}, \mathbf{x}') = \sum_{k=1}^{d} \beta_k \sum_{l=1}^{L-k+1} I(\mathbf{u}_{k,l}(\mathbf{x}) = \mathbf{u}_{k,l}(\mathbf{x}')).$$

Here, $\mathbf{u}_{k,l}(\mathbf{x})$ is the string of length k starting at position l of the sequence $\mathbf{x}$ and $I(\cdot)$ is the indicator function which evaluates to 1 when its argument is true and to 0 otherwise.

Definition at line 55 of file WeightedDegreeStringKernel.h.

Inheritance diagram for CWeightedDegreeStringKernel:



Constructor & Destructor Documentation

◆ CWeightedDegreeStringKernel() [4/4]

```
CWeightedDegreeStringKernel ( CStringFeatures< char > * l,
                              CStringFeatures< char > * r,
                              int32_t                   degree
                            )
```

constructor

Parameters

l	features of left-hand side
r	features of right-hand side
degree	degree

Definition at line **86** of file **WeightedDegreeStringKernel.cpp**.

Weighted degree kernel example

```
import shogun as sg
import numpy as np

dna_train = ['TTTCCC','TTTCCC','TTTTCC', 'TTTTTC', 'TTTTTT', 'ATTTTC']
labels_train = np.array([1,1,1,-1,-1,-1])
dna_test = ['TTTCCC','TTCCCC','TTTCCC']
labels_test = np.array([1,1,1])

features_train = sg.StringCharFeatures(dna_train, sg.DNA)
kernel_degree = 5
sk_train = sg.WeightedDegreeStringKernel(features_train, features_train, kernel_degree)
features_test = sg.StringCharFeatures(dna_test, sg.DNA)

# Train the Support Vector Machine
labels = sg.BinaryLabels(labels_train)
C = 1.0
svm = sg.LibSVM(C, sk_train, labels)
svm.train()

predicted_labels_train = svm.apply(features_train).get_labels()
print(predicted_labels_train)
predicted_labels_test = svm.apply(features_test).get_labels()
print(predicted_labels_test)

predicted_labels_train = svm.apply(features_train).get_values()
print(predicted_labels_train)
predicted_labels_test = svm.apply(features_test).get_values()
print(predicted_labels_test)
```

http://www.jmlr.org/papers/volume7/sonnenburg06a/sonnenburg06a.pdf

Journal of Machine Learning Research 7 (2006) 1531–1565 Submitted 10/05; Published 7/06

Large Scale Multiple Kernel Learning

Sören Sonnenburg SOEREN.SONNENBURG@FIRST.FRAUNHOFER.DE
Fraunhofer FIRST.IDA
Kekuléstrasse 7
12489 Berlin, Germany

Gunnar Rätsch GUNNAR.RAETSCH@TUEBINGEN.MPG.DE
Friedrich Miescher Laboratory of the Max Planck Society
Spemannstrasse 39
Tübingen, Germany

Christin Schäfer CHRISTIN.SCHAEFER@FIRST.FRAUNHOFER.DE
Fraunhofer FIRST.IDA
Kekuléstrasse 7
12489 Berlin, Germany

Bernhard Schölkopf BERNHARD SCHOELKOPF@TUEBINGEN.MPG.DE
Max Planck Institute for Biological Cybernetics
Spemannstrasse 38
72076, Tübingen, Germany

Multiple Kernel Learning

Multiple kernel learning (MKL) is based on convex combinations of arbitrary kernels over potentially different domains.

$$\mathbf{k}(x_i, x_j) = \sum_{i=1}^{K} \beta_k \mathbf{k}_i(x_i, x_j)$$

where $\beta_k > 0$, $\sum_{k=1}^{K} \beta_k = 1$, K is the number of sub-kernels, $\mathbf{k}$ is a combined kernel, $\mathbf{k}_i$ is an individual kernel and x_{ii} are the training data.

Regression is done by using CSVMLight. See Support Vector Regression for more details.

See [SRatschSchaferScholkopf06] for more information about MKL.

Example

Imagine we have files with training and test data. We create CDenseFeatures (here 64 bit floats aka RealFeatures) and CRegressionLabels as

```
features_train = RealFeatures(f_feats_train)
features_test = RealFeatures(f_feats_test)
labels_train = RegressionLabels(f_labels_train)
labels_test = RegressionLabels(f_labels_test)
```

Then we create indvidual kernels like CPolyKernel and CGaussianKernel which will be later combined in one CCombinedKernel.

```
poly_kernel = PolyKernel(10, 2)
gauss_kernel_1 = GaussianKernel(2.0)
gauss_kernel_2 = GaussianKernel(3.0)
```

Cell

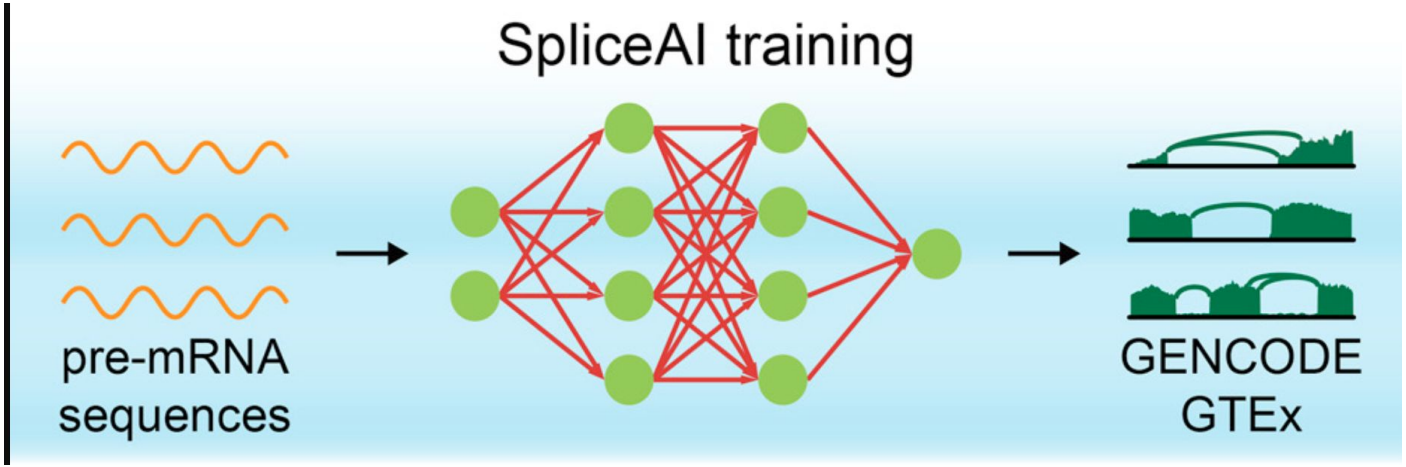
Volume 176, Issue 3, 24 January 2019, Pages 535-548.e24

Article

Predicting Splicing from Primary Sequence with Deep Learning

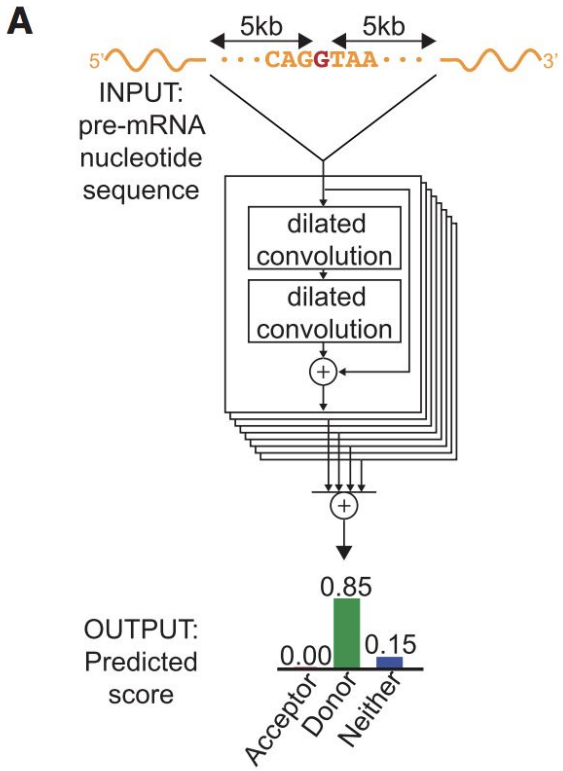
Kishore Jaganathan [1, 6], Sofia Kyriazopoulou Panagiotopoulou [1, 6], Jeremy F. McRae [1, 6], Siavash Fazel Darbandi [2], David Knowles [3], Yang I. Li [3], Jack A. Kosmicki [1, 4], Juan Arbelaez [2], Wenwu Cui [1], Grace B. Schwartz [2], Eric D. Chow [5], Efstathios Kanterakis [1], Hong Gao [1], Amirali Kia [1], Serafim Batzoglou [1], Stephan J. Sanders [2], Kyle Kai-How Farh [1, 7]

Overview



https://www.sciencedirect.com/science/article/pii/S0092867418316295#app2
https://en.wikipedia.org/wiki/RNA_splicing
https://www.cell.com/action/showPdf?pii=S0168-9525%2812%2900002-9

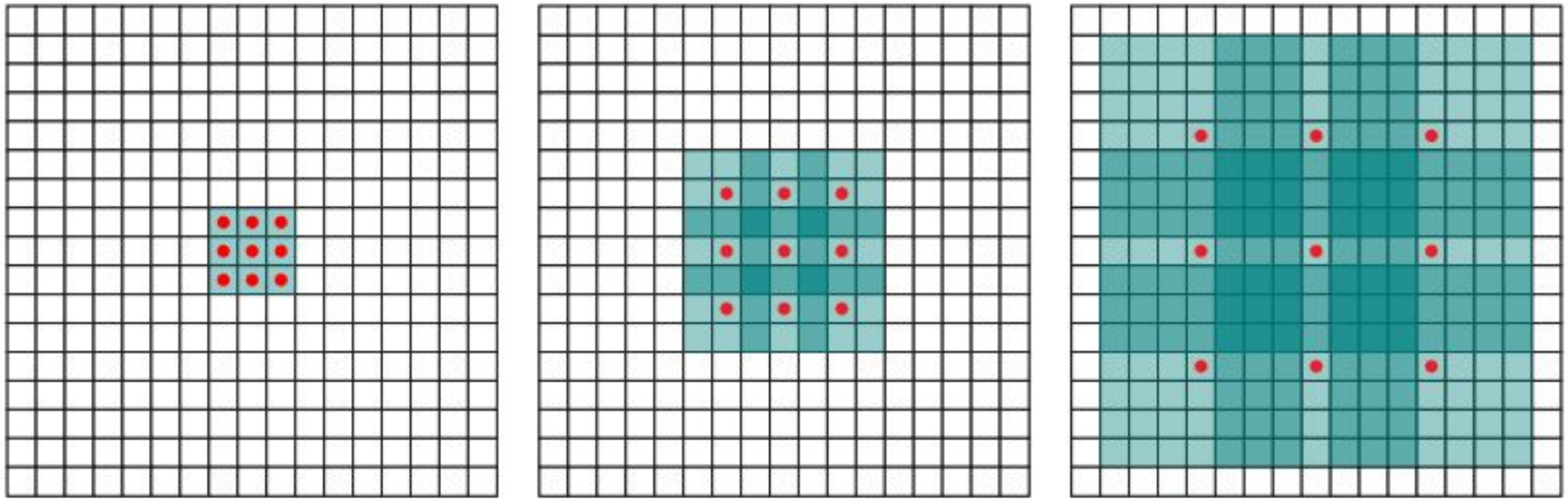
Model architecture



For each position in the pre-mRNA transcript, SpliceAI-10k uses 10,000 nucleotides of flanking sequence as input and predicts whether that position is a splice acceptor, splice donor, or neither

Uses GENCODE annotation as labeled data

Dilated convolution

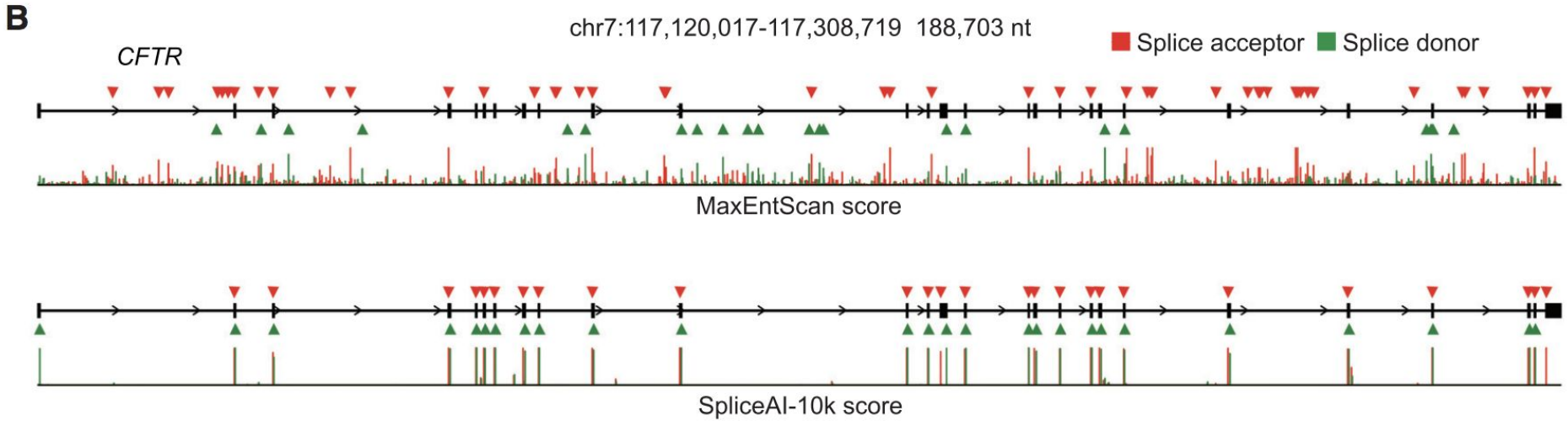


(a) (b) (c)

k=1 k=2 k=2

[Fisher Yu, Vladlen Koltun, ILCR 2016]

Example + comparison against MaxEntScan



The full pre-mRNA transcript for the CFTR gene scored using MaxEntScan (top) and SpliceAI-10k (bottom) is shown, along with predicted acceptor (red arrows) and donor (green arrows) sites and the actual positions of the exons (black boxes). For each method, we applied the threshold that made the number of predicted sites equal to the total number of actual sites.

To confirm that the network is not simply relying on exonic sequence biases, we also tested the network on long noncoding RNAs. Despite the incompleteness of noncoding transcript annotations, which is expected to reduce our accuracy, the network predicts known splice junctions in long noncoding RNAs (lincRNAs) with 84% top-k accuracy

E

	Top-*k* accuracy	PR-AUC
SpliceAI-80nt	0.57	0.60
SpliceAI-400nt	0.90	0.95
SpliceAI-2k	0.93	0.97
SpliceAI-10k	**0.95**	**0.98**
GeneSplicer	0.30	0.23
MaxEntScan	0.22	0.15
NNSplice	0.22	0.15

Effect of the size of the input sequence context on the accuracy of the network.

Top-k accuracy is the fraction of correctly predicted splice sites at the threshold where the number of predicted sites is equal to the actual number of sites present.

PR-AUC is the area under the precision-recall curve. We also show the top-k accuracy and PR-AUC for three other algorithms for splice-site detection.

Illumina / SpliceAI

Watch 6 | Star 48 | Fork 14

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Insights

A deep learning-based tool to identify splice variants

119 commits | 1 branch | 0 releases | 3 contributors | View license

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

Jaganathan v1.2.1: clean I/O handling, improved logs — Latest commit f6e289c 6 days ago

examples	v1.2.1: clean I/O handling, improved logs	6 days ago
spliceai	v1.2.1: clean I/O handling, improved logs	6 days ago
tests	v1.2: handles gvcf, bad inputs	10 days ago
.gitignore	v1.2.1: clean I/O handling, improved logs	6 days ago
COPYRIGHT.txt	Initial commit	4 months ago
LICENSE	Initial commit	4 months ago
README.md	v1.2.1: clean I/O handling, improved logs	6 days ago
setup.py	v1.2.1: clean I/O handling, improved logs	6 days ago

README.md

SpliceAI: A deep learning-based tool to identify splice variants

This package annotates genetic variants with their predicted effect on splicing, as described in Jaganathan *et al*, Cell 2019 in press.

Installation

The simplest way to install SpliceAI is through pip: