Hierarchical Classification Pre-processing Tool - User Guide

*Felipe Kenji Nakano (*fkenjinakano@gmail.com*)*

https://github.com/biomal/HCPT
September 10, 2018

Contents

1 Introduction **2**

2 Getting Started **2**

2.1 Installation and Dependencies 3

2.2 Input 4

3 Step by Step - Examples **7**

3.1 Example - 1 7

3.2 Example - 2 9

3.3 Example - 3 10

4 Adding Content **11**

4.1 Adding Sections and Attributes 11

5 Conclusion **13**

1 Introduction

This manual is a user guide to the open source framework Hierarchical Classification Pre-Processing Tool (HCPT). Regarding the ever-increasing amount of data being produced, HCPT is a tool designed to support Hierarchical Classification (HC) across applications. Similarly to traditional data classification, HC also builds a model to predict the class of a new instance, however, in this case, classes are structured according to a pre-defined hierarchy [5].

When compared to its traditional counterpart, HC brings several advantages. First, there are problems which are intrinsically hierarchical, such as transposable elements [4] and protein function prediction [1]. Furthermore, its results are more interpretative, since classifying an instance in a particular node provides information from its super and sub-classes. Finally, traditional classification problems can be converted to HC ones when appropriate, since it is possible to create an artificial hierarchy in accordance to the problem being addressed.

Despite HC's wide range of applications, there is still a lack of frameworks to support it, specially in the phase of data preparation. In this direction, HCPT aims to minimize such time-consuming process. By parsing a configuration file, it can create multiple datasets and map them to an user defined hierarchy. The framework facilitates research on HC, and makes it accessible to non-experts on machine learning.

The remainder of this manual is described as following: Section 2 presents instruction on how to get started with HCPT. Next, Section 3 brings examples which are easily adaptable to real case scenarios. Section 4 points out how to adapt and add context to our framework. Finally, Section 5 contains our conclusion and remarks for future works.

2 Getting Started

In this section, we provide an overview of HCPT. First we give an overview on how our framework works. Next we go through all the installation steps required to make a fresh UNIX system able to run HCPT. Finally, we also provide a description of every parameter available in the current version.

As portrayed in Figure 1, HCPT loads the the Settings (input) File alongside the Hierarchy File and Input Data. In this step, it parses the Settings file, checking for mistakes and reading the input data. Simultaneously, it also builds the hierarchy described in the Hierarchy File. Following, HCPT associates each label to its correspondent node in the hierarchy considering a numeric label.. Finally, features are extracted and written to the output file.

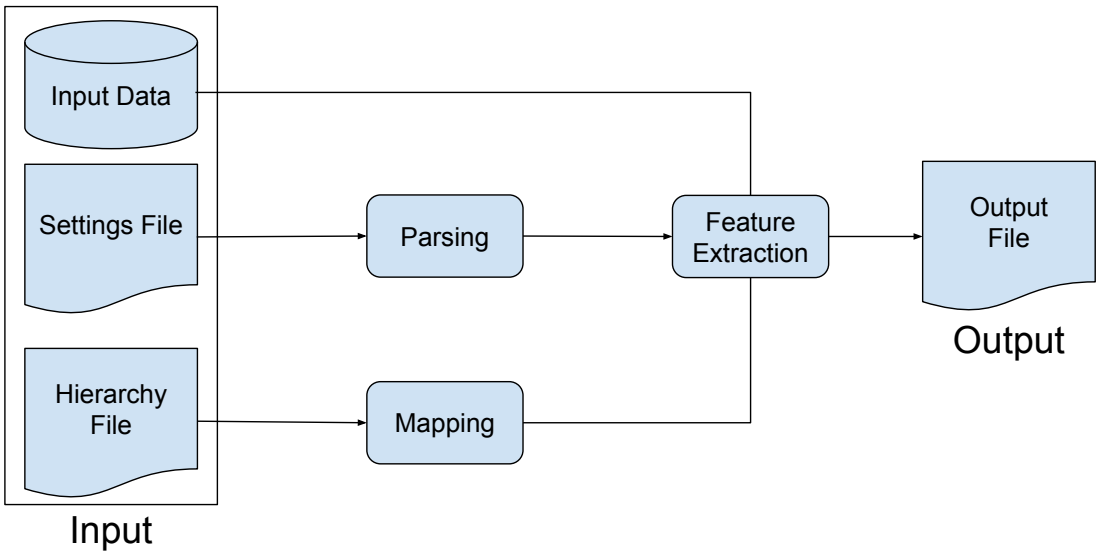

Figure 1: HCPT's work-flow diagram.

2.1 Installation and Dependencies

In order to get started, first download and unpack HCPT in a directory of your choice. Mind that our framework was entirely developed using Python 2.7 in an UNIX environment, and, up to the moment, we do not offer support to other operational systems. Following, it is necessary to install these dependencies:

- Networkx version 2.1: Library responsible to build the underlying structure and map all the hierarchical relationships;
- Pandas version 0.20.3: Library responsible to create the output .csv file;
- repDNA: Library responsible to extract features from the input data.

Considering a fresh installed UNIX system, the following commands will prepare your environment to run our framework.

First, we recommend using the pip tool[1] to easily install all the dependencies. Hence, run the following commands lines:

```
sudo apt-get update
sudo apt-get install python-pip
```

Next, use pip to install Networkx, pandas and repDNA. Naturally, pip will also install these packages' dependencies.

```
sudo pip install networkx==2.1

sudo pip install pandas==0.20.3
```

[1] https://pypi.org/project/pip/

```
sudo pip install repDNA
```

Note that this is the vanilla version of our framework. For better user experience, please stick to the versions specified. The framework is not guaranteed to work if its dependencies versions are mismatched.

Also notice that the repDNA library is employed [3], as it offers multiple ways of feature extracting to genomics, including our original application, Transposable Elements. Nonetheless, other libraries can be added, as detailed in Section 4.

2.2 Input

As input to our framework, we have created a Settings file where the user specify all the input parameters. To make it more interpretative, it is split into Sections which are further divided into Attributes. To clarify all of its components, we present a dummy file in Table 1.

Following, we bring a description of every Attribute:

- Verbose = 0 means silent mode, whereas 1 produces complementary text
- Input = path to the Input file, as exemplified in Table 2
- Output = path to the Output file to be generated
- Type = Type of feature to be extracted.
- K = parameter to {Kmer,Reverse,PSEKNC}
- Normalize = parameter to {Kmer,Reverse}. If True features will be normalized from 0 to 1
- UpTo = parameter to {Kmer,Reverse}, if True Kmer or Reverse up to the value of K will be extracted
- Lamada = parameter to {PSEDNC,PSEKNC,PCPSEDNC,PCPSETNC, SCPSEDNC,SCPSETNC}
- W = parameter to {PSEDNC,PSEKNC,PCPSEDNC,PCPSETNC,SCPSEDNC, SCPSETNC}
- AllProperty = parameter to {PSEDNC,PSEKNC,PCPSEDNC,PCPSETNC, SCPSEDNC,SCPSETNC,DAC,DCC,DACC,TAC,TCC,TACC}. This Attribute overwrites PhycheIndex and ExtraIndex.
- PhycheIndex = parameter to {DAC,DCC,DACC,TAC,TCC,TACC}
- ExtraIndex = parameter to {DAC,DCC,DACC,TAC,TCC,TACC}
- Lag = parameter to {DAC,DCC,DACC,TAC,TCC,TACC}
- File = path to Hierarchy File, as exemplified in Table 3

In this version, all the features are incorporated straight from the repDNA library. Features can be divided in three groups: Nucleic Acid Composition, AutoCorrelation and Pseudo Nucleic Acid Composition. As its name suggests, the first one concerns features related exclusively to the placement of the sequence's aminoacids, and its features include Kmers and Reverse Kmer. Similarly, the second group

Table 1: All Input Parameters

		#Mandatory/Default	#Values
[General]			
Verbose	=	0	int: {0,1}
[Data]			
Input	=	Mandatory	String: path to file
Output	=	Mandatory	String: path to file
[Features]			
Type	=	Mandatory	String:{Kmer,Reverse,PSEDNC,PSEKNC,PCPSEDNC, PCPSETNC,SCPSEDNC,SCPSETNC,DAC,DCC, DACC,TAC,TCC,TACC}
K	=	4	int: higher than 0
Normalize	=	False	boolean:{True,False}
Lamada	=	1	int
UpTo	=	1	int
W	=	0.005	float
AllProperty	=	True	boolean:{True,False}
Lag	=	1	int
PhycheIndex	=	empty	list
ExtraIndex	=	empty	list
[Hierarchy]			
File	=	Mandatory	String: path to file

presents features which measure the correlation between two properties within a sequence, and the features types DAC, DCC, DACC, TAC, TCC and TACC are placed in this group . Lastly, Pseudo Nucleic Acid Composition represents features considering both local and global aspects in terms of DNA composition, and include the remaining features PSEDNC, PSEKNC, PCPSEDNC, PCPSETNC, SCPSEDNC and SCPSETNC.

Recall that our focus is not to handcraft new features, but to provide a framework to support HC. For further description on their mathematical formulation and rationale, the interested reader can refer to the repDNA's manual[2].

Since the original application is related to genomics, the "Input" Attribute in the "Data" Section should be in the FASTA format. As seen in Figure 2, this format is composed of a two line entry per sequence. Marked with a ">" character in its beginning, the first line contains the identification of the sequence, along with extra information. In our framework, it should contain its usual description, alongside the character "|" and the name of a node specified in the Hierachy File. Likewise, since the Output file is supposed to be used as input to classification algorithms, the standard format is a .csv file (comma separated value).

The Hierarchy file should contain all the nodes in the hierarchy with their respective names. Taking the file displayed in Table 3, our framework will build the Hierarchy portrayed in Figure 2. Mind that, the nodes are described by a numerical scheme, where each number corresponds to a node in a particular level, and the character "." (point) is a level separator. Also notice that the framework will automatically build all the nodes in the path of an particular node. Hence, there is no need to place them in order in the file.

Table 2: Input Fasta File

sequence1|miRNA
AGCCGAGGAC
sequence2|foldedDNA
CTAGTGTTTATG
>sequence3|RNA
GAGGCCACACAA

[2] https://usermanual.wiki/Pdf/repDNAmanual.947194590/view

Figure 2: Dummy Hierarchy

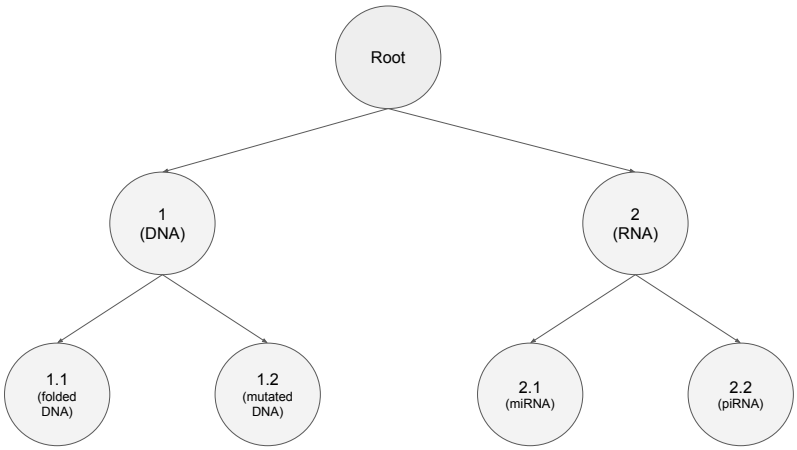

Table 3: Hierarchy File Example

DNA	=	1
foldedDNA	=	1.1
mutatedDNA	=	1.2
RNA	=	2
miRNA	=	2.1
piRNA	=	2.2

3 Step by Step - Examples

In this section, we go through three examples, covering different perspective and groups of features. All the data used here is fictional, and do not represent real DNA sequences.

After downloading HCPT, run the main file passing your Settings File as the first parameter. The following command line exemplifies it:

```
python main.py settings.s
```

Recall that, there is no need to manually build the hierarchy, since HCPT automatically does it in accordance with the Hierarchy File provided.

3.1 Example - 1

In the first example (Table 4), we cover how to extract Kmer with K = 2. In the field of bioinformatics, K-mers is a popular feature,

with its usage related to genome and transcriptome assembly, and metagenomic sequencing. In plain English, K-mers are quantifications of subsequences of size K. Taking our example into account, the occurrences of $(AA, AT, AG, AC....CC)$ are counted. Comparatively, larger values for K generate more features. However, a very high number of features may include irrelevant features, as their frequency tend to be low, and many zeros will appear.

Table 4: Kmer Example

[General]		
Verbose	=	0
[Data]		
Input	=	input.fasta
Output	=	kmers.csv
[Features]		
Type	=	Kmer
K	=	2
Normalize	=	False
UpTo	=	True
[Hierarchy]		
File	=	hierarchy.txt

This Settings file could be used for the Reverse K-mers. In order to do so, "Type = Kmer" needs to be replaced by "Type = Reverse". Similarly to K-mer, its reverse version also counts subsequences, however, reversed subsequences are summed up together. Hence, AGC and GCT are added up as the same feature. When compared to regular Kmers, Reverse is a more compact, and even more meaningful in some cases, since the number of zeros is likely to reduce due to its cumulative nature.

In both cases, features will be presented as their raw number, since "Normalize = False". Furthermore, K-mers of size 1 will also be con-

catenated, seeing that “UpTo = True”.

3.2 Example – 2

Table 5: DAC Example

[General]		
Verbose	=	0
[Data]		
Input	=	input.fasta
Output	=	dac.csv
[Features]		
Type	=	DAC
AllProperty	=	True
Lag	=	1
[Hierarchy]		
File	=	hierarchy.txt

In the second example, we present how to extract the AutoCorrelation features. These features measure the correlation between any two properties, hence for such task, it is necessary to specify what properties should be considered. Although the example in Table 5 is displayed with the parameter “DAC”, it is also applicable to DAC,DCC,DACC,TAC,TCC,TACC. The following list provides their full nomenclature:

- DAC: Dinucleotide-based auto covariance;
- DCC: Dinucleotide-based cross covariance;
- DACC: Dinucleotide-based auto-cross covariance;
- TAC: Trinucleotide-based auto covariance;
- TCC: Trinucleotide-based cross covariance;
- TACC: Trinucleotide-based auto-cross covariance.

The first type, DAC, measures the correlation of a property (physico-chemical index) between two dinucleotides which are basically subsequences of size 2. Likewise, DCC also measures the correlation between two dinucleotides, however it considers only different indexes. Next, DACC simply combines DAC and DCC. In the case of the trinucletide-based features, the same logic applies, notwithstanding trinucleotides are evaluated.

As mentioned in Section 2, it is possible to go for all properties by setting the parameter, "AllPropert = True". In case of specific properties, one can set "PhycheIndex = [Twist]" if Twist is desired. For the list with all properties, please consult repDNA's manual [3]. Moreover, user customized indexes are also allowed using the parameter "ExtraIndex".

3.3 Example – 3

Table 6: PSEDNC Example

[General]		
Verbose	=	0
[Data]		
Input	=	input.fasta
Output	=	psednc.csv
[Features]		
Type	=	PSEDNC
Lamada	=	3
W	=	0.005
[Hierarchy]		
File	=	hierarchy.txt

In this last example, we detail how to extract Pseudo Acid Nucleic Composition features. Originally proposed by Chou [2], these features have proved to be meaningful in many bioinformatics contexts.

[3] https://usermanual.wiki/Pdf/repDNAmanual.947194590/view

In a nutshell, this kind of features incorporates local and global notions, since aminoacids which are placed either close or far are taken in consideration, unlikely the example from Table 4, where features only reflects local aspects.

In the current version, six types of Pseudo Acid Nucleic Composition are available. The following list presents their full names.

- PSEDNC: Pseudo dinucleotide composition;
- PSEKNC: Pseudo k-tupler composition;
- PCPSEDNC: Parallel correlation pseudo dinucleotide composition;
- PCPSETNC: Parallel correlation pseudo trinucleotide composition;
- SCPSEDNC: Series correlation pseudo dinucleotide composition;
- SCPSETNC: Series correlation pseudo trinucleotide composition.

In all of them, the local aspects are incorporated by counting dinucleotides throughout the sequence. Likewise, the global factor is measured using physicochemical indexes, similarly to AutoCorrelation features. Hence, pseudo features are created by combining both of them, and the methods differ on how they to it.

4 Adding Content

Considering that this is the very first version of HCPT, and the fact that HC covers a considerable number of applications, expanding our framework is essential and easy to do. In order to do so, next we present how to add content to HCPT.

4.1 Adding Sections and Attributes

As showed in Table 1, our settings file is divided into sections that contain Attributes. Sections should be named after a general concept. For instance, the "Data" section describes all attributes related to the input and output data. Similarly, Attributes should point out to an specific aspect, like the Input Attribute.

To add new Sections in the settings File, follow these steps:

1. Create a new Section in the Settings.py file;
2. Add the new Section to the "sectionsName" list in the Parser.py file.

In the Settings.py file, more specifically in the "initializeDefault" method, the new Section should be instantiated, and Attributes added to it. Following, an example on how to do it.

```
section = Section('Example')
section.addAttribute('NewIntAttribute',0)
```

```
section.addAttribute('NewStringAttribute','')
section.addAttribute('NewFloatAttribute',0.0)
section.addAttribute('NewListAttribute',[])
self.addSection(section)
```

Initially, a new Section is created with the dummy name “Example”. Then, four new attributes are added to it. Every attribute is instantiated using two parameters: Name and Default Value. Naturally, Name refers to how it should be accessed in the Settings File, whereas Default Value represents its value when none is passed. Recall that, if you only want to add an extra Attribute, there is no need to create a new Session.

After creating Sections and Attributes, we proceed to modify HCPT’s behaviour. In the Parser.py file, look for the method “run”. In this method, every major aspect of HCPT, such as Hierarchy, Feature Extractor, Input and Output, is handled.

In order to access values from the Settings file, look for the corresponding method.

```
Section = getSection('sectionName')
Attribute = getAttribute('attributeName')
```

Remind that every Section is placed in the Settings object, whereas Attributes are allocated in their own Section. Hence, to handle the Attribute “NewIntAttribute”, in Section “Example”, use the following commands.

```
exampleSection = self.settings.getSection('Example')
newAttribute = exampleSection.getAttribute('NewIntAttribute')
newAttributeValue = newAttribute.value
defaultOrValue = newAttribute.enabled
```

In the “newAttributeValue” variable, the value from the Settings is stored. When a value is passed via Settings file, the variable “defaultOrValue” will be True, and False in the opposite situation.

Also, in case of creating a mandatory Attribute, add it to the dictionary “mandatoryAttributes” in the Parser.py file. Moreover, consider creating a variable in Settings.py to store the default value of the new Attribute.

After being able to add Sections and Variables, one may want to handle possible mistakes such as invalid values or attesting the presence of mandatory values. In order to facilitate debugging, HCPT has built-in exceptions. Up to moment, four of them are available:

- MissingAttributeException: Attribute given in Settings file not declared in Settings.py;
- MissingMandatoryValueException: Mandatory Attribute was not given a value in Settings file;

- MissingSectionException: Section given in Settings file not declared in Settings.py;
- WrongValueException: Value given for an Attribute does not match its type.

5 Conclusion

In this user guide, we have provided guidelines on how to install and use our framework, HCPT. By allowing feature extraction and hierarchical mapping, it focus on reducing the amount of manual work required to study HC. It also makes HC more accessible to non-machine learning experts, since its parametrization is done intuitively.

As future works, we plan to extend our framework. As discussed by Silla [5], HC is performed using different approaches. Subsequent versions should include the task of classification as well, building up to a complete framework. Moreover, HC has been applied to other domains, such as medical images, audio and text classification. Hence, more features are necessary to correctly address these other applications.

References

[1] Ricardo Cerri, Rodrigo C. Barros, André C. P. L. F. de Carvalho, and Yaochu Jin. Reduction strategies for hierarchical multi-label classification in protein function prediction. *BMC Bioinformatics*, 17(1):373, 2016.

[2] Kuo-Chen Chou. Prediction of protein cellular attributes using pseudo-amino acid composition. *Proteins: Structure, Function, and Bioinformatics*, 43(3):246–255, 2001.

[3] Bin Liu, Fule Liu, Longyun Fang, Xiaolong Wang, and Kuo-Chen Chou. repdna: a python package to generate various modes of feature vectors for dna sequences by incorporating user-defined physicochemical properties and sequence-order effects. *Bioinformatics*, 31(8):1307–1309, 2014.

[4] F. K. Nakano, W. J. Pinto, G. L. Pappa, and R. Cerri. Top-down strategies for hierarchical classification of transposable elements with neural networks. In *International Joint Conference on Neural Networks (IJCNN)*, pages 2539–2546, 2017.

[5] Jr. Silla, CarlosN. and AlexA. Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22(1-2):31–72, 2010.