ErrorX Documentation

Release 1.1.0

Alex Sevy

Apr 13, 2020

CONTENTS

CHAPTER

ONE

OVERVIEW

ErrorX is a software package for correction of DNA sequencing errors. In next-generation sequencing, it's common that many of the output sequences are actually errors, different from the input DNA sequence. This can be caused by the way the DNA was prepared, amplification before sequencing, and errors introduced by the sequencing machine itself. Unfortunately, these errors can be very costly to a next-generation sequencing pipeline, as you can never be sure which sequences are true and which are mistakes.

ErrorX solves this problem by using deep neural networks to predict sites where an error has been introduced. Trained on a dataset of tens of millions of sequences, ErrorX can pinpoint with extremely high accuracy positions where an error was likely introduced, saving valuable time and energy by removing these bases from further analysis. In benchmark studies, ErrorX was able to remove up to 36% of errors from a dataset with a false positive rate of less than 0.05%.

ErrorX currently supports processing of antibody and T-cell repertoire sequencing data from human and mice, gathered on Illumina HiSeq and MiSeq instruments.

CHAPTER

TWO

INSTALLATION

To install ErrorX, please visit the Releases page on the ErrorX Github account. Once you have downloaded the zipped archive for your operating system (this will have the extension `.tar.gz`), you are ready to unpack the folder and starting running.

Run the following command to extract the files from the archive:

```
tar xvfz ErrorX-1.1.0_mac.tar.gz
```

You can also use a third-party software such as 7-zip to extract the files.

The ErrorX binary is ready to go from there - it is located at `ErrorX/bin/errorx`

Note: the binary relies on data contained in the ErrorX folder. For best results, leave the directory structure of ErrorX intact.

CHAPTER

THREE

QUICKSTART

After installing ErrorX you are ready to run your first sequences. You can find example sequences to use as an example in the `ErrorX/documentation/` folder. Use the following command to run ErrorX prediction:

```
ErrorX/bin/errorx --species human --format fastq --out ErrorX_out.tsv
--allow-nonproductive ExampleSequences.fastq
```

You will find a tab-separated file called `ErrorX_out.tsv` with all the annotated sequences and their predicted errors.

CHAPTER

FOUR

INPUT

4.1 FASTQ

The most common way to run ErrorX is with a FASTQ file. ErrorX will run germline assignment on the sequences and use sequence data and quality information to feed into the error prediction model.

4.2 FASTA

Alternatively, ErrorX can be run on a FASTA file. However, please note that error correction cannot be performed with FASTA input, since error correction requires the quality information that is missing in FASTA format. Running ErrorX on FASTA input will simply run germline assignment and annotate the input sequences.

4.3 TSV

Alternatively you can provide sequences in TSV format. If you've already run germline assignment with another software, you can save time and only run error prediction in ErrorX. TSV files should have four columns, separated by tabs, with no header:

1. Sequence ID
2. Nucleotide sequence
3. Inferred germline sequence
4. PHRED score.

4.4 Output

The output of ErrorX is a TSV file summarizing the input sequences along with a corrected nucleotide sequence, where the predicted errors are replaced by 'N'. If you input a FASTQ sequence, then the TSV will have information on the V, D, and J genes, as well as the level of somatic mutation and CDR3 sequence.

4.5 Full options list

Below is a list of all options that can be given to the application:

```
-h [ --help ]                                          Produce help message

-f [ --format ] arg                                   Input file format. Valid entries␣
↪are fastq, fasta, or tsv.

-o [ --out ] arg (=out.tsv)                Output file (Default=out.tsv)

-s [ --species ] arg (=human)         Species for IGBLAST search. Valid entries are␣
↪human or mouse (Default=human)

--igtype arg (=Ig)                                    Receptor type for IGBLAST search.␣
↪Valid entries are Ig or TCR. (Default=Ig)

-n [ --nthreads ] arg (=-1)                Number of threads to use during execution.␣
↪Enter -1 to use all available (Default=-1)

-e [ --error-threshold ] arg (=0.730736)        Probability cutoff for a base to be␣
↪considered an error. Higher=more stringent in calling errors. Don't change this␣
↪value unless you know what you are doing.

--infile arg                                           Input file name

--version                                                  Print ErrorX version␣
↪information and exit

-v [ --verbose ] arg (=1)                Verbosity level: should ErrorX output extra␣
↪warnings and messages? 0: don't output any message at all. 1: output progress␣
↪during processing. 2: output progress and debugging messages. (default=1)

--allow-nonproductive                          Allow nonproductive and out-of-frame␣
↪sequences to be included? (default=No)

 --license arg                                          License key to activate full␣
↪version of ErrorX
```

CHAPTER

FIVE

C++ API

5.1 Using the API

In addition to the ErrorX binary, you can integrate the functions into your own C++ application using the shared library API. The dynamic library is location in the `ErrorX/lib/` folder. In addition, all of the input files used in this tutorial can be found in the `ErrorX/documentation` folder.

5.2 Walkthrough

Below is a sample of code showing how you would use ErrorX in a C++ application. In this example, we are going to process a file called `test.tsv`, and output it to `test_out.tsv`. ErrorX takes TSV files with four columns: the sequence ID or name, the nucleotide sequence itself, the inferred germline sequence, and the PHRED score from the sequencer.

First, we need to include the relevant ErrorX headers:

```
#include "errorx.hh"
#include "SequenceRecords.hh"
#include "ErrorXOptions.hh"
```

Next, we will create an ErrorXOptions object that contains all the options related to your processing. We will set the input filename to test.tsv, and the file type to TSV. ErrorX currently accepts files in FASTQ, FASTA, TSV format. We will also set the species as human - ErrorX currently supports human and mouse. Lastly, we need to set the path to the ErrorX folder. This will be used to locate different libraries needed for execution.

```
errorx::ErrorXOptions options( "test.tsv", "tsv" );
options.outfile( "test_out.tsv" );
options.errorx_base( "/path/to/ErrorX/" );
options.species( "human" );
```

Now, we're ready to run the protocol. If we have provided a FASTQ or FASTA file, ErrorX will go through germline assignment and then error correction. If we use a TSV file with the germline sequence already specified, then it will only do error correction:

```
errorx::run_protocol_write( options );
```

This will run the protocol and output the results to our previously defined outfile.

You can also run a query using a FASTQ file, and ErrorX will do the germline assignment for you. To do so, you only need to give a FASTQ file to ErrorXOptions, and set the format type to `"fastq"`, as shown below:

```
errorx::ErrorXOptions options( "test.fastq", "fastq" );
options.outfile( "test_out.tsv" );
options.species( "human" );
options.errorx_base( "/path/to/ErrorX/" );
errorx::run_protocol_write( options );
```

5.3 Viewing and Analyzing SequenceRecords

The previous example only showed how to run the protocol and output the results to a file. But, you can also directly get the results of error correction to do further analysis within your C++ application. The central object in ErrorX is a SequenceRecord - this holds all the information for one nucleotide sequence, including its germline genes, level of somatic hypermutation, chain type, and eventually its corrected sequence. There is also a SequenceRecords object, which is a collection of separate SequenceRecord objects that have been processed together. Using the SequenceRecords object we can get access to the overall error rate in our sample, individual corrected sequences, and more. Using the ErrorXOptions object that we created before, we can get a pointer to a SequenceRecords object will all the information in the output file:

```
errorx::SequenceRecords* records = errorx::run_protocol( options );
```

To work with the original and corrected nucleotide sequences of one of the entries, we can access them directly through the get() method. This will return a pointer to the SequenceRecord at the index that we provide:

```
std::cout << "Original NT sequence: " << records->get( 0 ).full_nt_sequence() <<␣
↪std::endl;
std::cout << "Corrected NT sequence: " << records->get( 0 ).full_nt_sequence_
↪corrected() << std::endl;
```

We can also print out a summary of the data in our SequenceRecords object:

```
records->print_summary();
```

A full working example is shown below:

```
#include "errorx.hh"
#include "SequenceRecords.hh"
#include "ErrorXOptions.hh"
#include <iostream>

int main() {
        errorx::ErrorXOptions options( "test.tsv", "tsv" );
        options.outfile( "test_out.tsv" );
        options.species( "human" );
        options.errorx_base( "/path/to/ErrorX/" );
        errorx::run_protocol_write( options );

        errorx::SequenceRecords* records = errorx::run_protocol( options );
        std::cout << "Original NT sequence: " << records->get( 0 )->full_nt_
↪sequence() << std::endl;
        std::cout << "Corrected NT sequence: " << records->get( 0 )->full_nt_sequence_
↪corrected() << std::endl;
                        records->print_summary();
                }
```

5.4 Compilation

To use the ErrorX library in a C++ application, you only need to include two extra steps while you are compiling your code: 1. include the header files from ErrorX, and 2. dynamically link to the library. A successful clang command looks something like this:

```
clang++ -std=c++11 -IErrorX/include/ -LErrorX/lib/ -lerrorx ExampleApp.cc
```

The `-I` flag will include all of the ErrorX headers during compilation, and the `-L` and `-l` flags will makes sure the correct library is linked.

Note: when you run the application linking to ErrorX, the ErrorX library **must** be in the path where your OS is going to look for dynamically linked libraries. In Linux this path is set by an environmental variable called `LD_LIBRARY_PATH`, or `DYLD_LIBRARY_PATH` on Mac. Make sure to run the following command to set the path correctly (or, even better, add this line to your `~/.bashrc` and never worry about it again!):

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:ErrorX/lib/
```

CHAPTER

SIX

PYTHON API

6.1 Installation

The Python packages are also located in the folder you just extracted. ErrorX supports Python versions 2.7 and 3.6. To install the Python 2.7 package simply run

```
pip install python2_bindings/
```

and for version 3.6 run

```
pip3 install python3_bindings/
```

You are now ready to run ErrorX through the Python interface!

6.2 Walkthrough

To run the Python interface, you first need to import the errorx package to have access to all the functions to do data correction. You can also import the ErrorXOptions class if you want to have more fine-tuned control over the settings. To run the entire protocol, you need to provide an input file, as well as its format. A full example is provided in `ErrorX/documentation/ExampleApp.py`.

```
import errorx as ex
from errorx import ErrorXOptions

options = ErrorXOptions('example.fastq','fastq')
options.outfile( 'example_out.tsv' )
options.species( 'mouse' )
options.igtype( 'Ig' )
ex.run_protocol( options )
```

You can also get corrected sequences from within the Python script that you can then operate on, shown below. This function takes in lists of nucleotide sequences, their inferred germline sequences, and the PHRED quality scores, and returns a list of corrected sequences corresponding to the input sequences:

```
sequences = ['TACTCCCG']
germline_sequences = ['TACTCCCA']
phred_scores = ['HEIHHIII']

results = correct_sequences(sequences, germline_sequences, phred_scores, options )
```

You can also input a single sequence, and get the predicted probability of error for each position along that sequence (positions are zero-indexed):

```
sequence = 'TACTCCCG'
germline_sequence = 'TACTCCCA'
phred_scores = 'HEIHHIII'

probabilities = get_predicted_errors(sequences, germline_sequences, phred_scores,␣
↪options )

# returns probability of error at position 0
print probabilities[0]
```

CHAPTER

SEVEN

JAVA API

7.1 Installation

The ErrorX Java library is also located in the gzipped archive, in the folder java_bindings. To install you don't need to do anything special - all the functionality is contained in the `ErrorX.jar` archive within the `java_bindings` folder. When you are compiling your Java application simply add the flag

```
-cp 'ErrorX/java_bindings/ErrorX.jar'
```

to make sure the JAR is linked, and you are all set!

Note: the JAR relies on data files contained in the java_bindings folder. For best results, leave the directory structure of java_bindings intact and only link the JAR. Alternatively, if you have to move the JAR file, make sure to move all the other directories in `java_bindings` along with it.

7.2 Walkthrough

Similar to the Python interface, the Java API consists of three main functions:

1. Runs the whole protocol on a FASTQ or TSV file and writes to another TSV file,
2. Takes in an array of sequences, germline sequences, and PHRED scores and returns an array of corrected sequences,
3. Takes in a single sequence, germline sequence, and PHRED score and returns an array where element i is the probability that nucleotide at position i is an error (positions are zero-indexed).

Below are examples of how to run each scenario. A full example is presented in `ErrorX/documentation/ExampleApp.java`. First import the errorx package.

```
import errorx.ErrorX;
```

Next, in the body of your class, you need to create an ErrorXOptions object that defines, at minimum, the name of the file to be processed and its format (either FASTQ or TSV). ErrorXOptions can also be used for more advanced control of the protocol.

```
ErrorXOptions options = new ErrorXOptions( "example.tsv", "tsv" );
options.outfile( "out.tsv" );
options.species( "mouse" );

ErrorX ex = new ErrorX();
ex.runProtocol( options );
```

For scenario two, you need to make arrays of Strings representing the nucleotide sequence, germline sequence, and PHRED score of your query. The function `correctSequences` will return an array of Strings with the corrected nucleotide sequences corresponding to the input:

```
String sequence = "TACTCCCG";
String germline_sequence = "TACTCCCA";
String phred_score = "HEIHHIII";

String[] sequences = new String[1];
sequences[0] = sequence;

String[] germline_sequences = new String[1];
germline_sequences[0] = germline_sequence;

String[] phred_scores = new String[1];
phred_scores[0] = phred_score;

String[] correctedSequences = new ErrorX().correctSequences( sequences, germline_
↪sequences, phred_scores );
```

For the final scenario, you need to input a single sequence, germline, and PHRED score. The function `getPredictedErrors` will return an array of doubles representing the probability of error at each position (zero-indexed):

```
double[] errorPredictions = new ErrorX().getPredictedErrors(
sequence, germline_sequence, phred_score );

System.out.println( errorPredictions[0] ); // prints the probability of error at␣
↪position 0
```

CHAPTER

EIGHT

ERRORXOPTIONS

All the options surrounding the ErrorX protocol are controlled by a class called ErrorXOptions. This has the same function in the C++, Python, and Java APIs. The options that you can control with this class are the same as the command-line options. Please see the documentation on full options list above for full details on which options can be controlled.