ErrorX User Guide

Contents

Overview

ErrorX is a revolutionary software 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 identify errors with an accuracy of 99.9% and a false positive rate of only 0.1%.

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

Installation

To install ErrorX, you need to request a download link from your Endeavor Bio representative. They will give you a link to the appropriate download based on your operating system. Once you have downloaded the zipped archive (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-1.0_mac.tar.gz
```

The ErrorX binary is ready to go from there - it is located at `Errorx-1.0/bin/errorx`

Note: the binary relies on data contained in the `ErrorX-1.0` folder. For best results, leave the directory structure of ErrorX intact.

Quickstart

After installing ErrorX you are ready to run your first sequences. Use the following command to run ErrorX prediction:

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

This will create a file called `ExampleSequences.tsv` that contains a summary of your input sequences, plus the nucleotide sequences with ErrorX correction applied.

Input

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.

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.

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.

C++ API

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-1.0/lib/` folder.

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 or 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 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 );
```

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();
    }
```

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/
```

Python API

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!

Java API

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-1.0/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.