Discovering Information Relevant to API Elements Using Text Classification

Gayane Petrosyan

November 6, 2013

API Usability

Java SE 6
3774 classes, 203 packages

YourExtraLife
104 third party libraries

API Learning

Developers get knowledge from a lot of different resources

- API documentation (e.g. Javadoc)
- API tutorials (both official and not official)
- discussion forums/mailing lists
- source codes/code snippets

API tutorials

How to find useful information about an API type in API tutorial?

- Browse table of content, if there is any?
  - “Next”, “Example”, “Overview”, “General case”, etc.
- Search API type as a word?

JodaTime - Section "Instants"

Instants

The most frequently used concept in Joda-Time is that of the *instant*. An Instant is defined as *a moment in the datetime continuum specified as a number of milliseconds from 1970-01-01T00:00Z*. This definition of milliseconds is consistent with that of the JDK in `Date` or `Calendar`. Interoperating between the two APIs is thus simple.

Within Joda-Time an instant is represented by the `ReadableInstant` interface. The main implementation of this interface, and the class that the average API user needs to be most familiar with, is `DateTime`. DateTime is immutable - and once created the values do not change. Thus, this class can safely be passed around and used in multiple threads without synchronization.

The millisecond instant can be converted to any date time field using a *Chronology*. To assist with this, methods are provided on `DateTime` that act as getters for the most common date and time fields.

We discuss the chronology concept a litte further on in this overview.

A companion mutable class to `DateTime` is `MutableDateTime`. Objects of this class can be modified and are not thread-safe.

Other implementations of `ReadableInstant` include `Instant` and `DateMidnight`.

JodaTime - Section "Instants"

Instants

The most frequently used concept in Joda-Time is that of the *instant*. An Instant is defined as *a moment in the datetime continuum specified as a number of milliseconds from 1970-01-01T00:00Z*. This definition of milliseconds is consistent with that of the JDK in `Date` or `Calendar`. Interoperating between the two APIs is thus simple.

Within Joda-Time an instant is represented by the `ReadableInstant` interface. The main implementation of this interface, and the class that the average API user needs to be most familiar with, is `DateTime`. DateTime is immutable - and once created the values do not change. Thus, this class can safely be passed around and used in multiple threads without synchronization.

The millisecond instant can be converted to any date time field using a *Chronology*. To assist with this, methods are provided on `DateTime` that act as getters for the most common date and time fields.

We discuss the chronology concept a litte further on in this overview.

A companion mutable class to `DateTime` is `MutableDateTime`. Objects of this class can be modified and are not thread-safe.

Other implementations of `ReadableInstant` include `Instant` and `DateMidnight`.

Problem Description

Discovering relevant sections of an API tutorial to help programmers to find additional related information about API elements they are interested in

- API Element Identification
- API tutorial Segmentation
- Classification of relevant and not relevant cases

Finding API Elements- Recodoc

Chat

A chat creates a new thread of messages (using a thread ID) between two users. The following code snippet demonstrates how to create a new Chat with a user and then send them a text message:

```
// Assume we've created a Connection name "connection".
ChatManager chatmanager = connection.getChatManager();
Chat newChat = chatmanager.createChat("jsmith@jivesoftware.com", new MessageList
    public void processMessage(Chat chat, Message message) {
        System.out.println("Received message: " + message);
    }
});

try {
    newChat.sendMessage("Howdy!");
}
catch (XMPPException e) {
    System.out.println("Error Delivering block");
}
```

The `Chat.sendMessage(String)` method is a convenience method that creates a Message object, sets the body using the String parameter, then sends the message. In the case that you wish to set additional values on a Message before sending it, use the `Chat.createMessage()` and `Chat.sendMessage(Message)` methods, as in the following code snippet:

```
Message newMessage = new Message();
newMessage.setBody("Howdy!");
message.setProperty("favoriteColor", "red");
newChat.sendMessage(newMessage);
```

Figure : Smack - Section “Chat”

Tutorial Segmentation

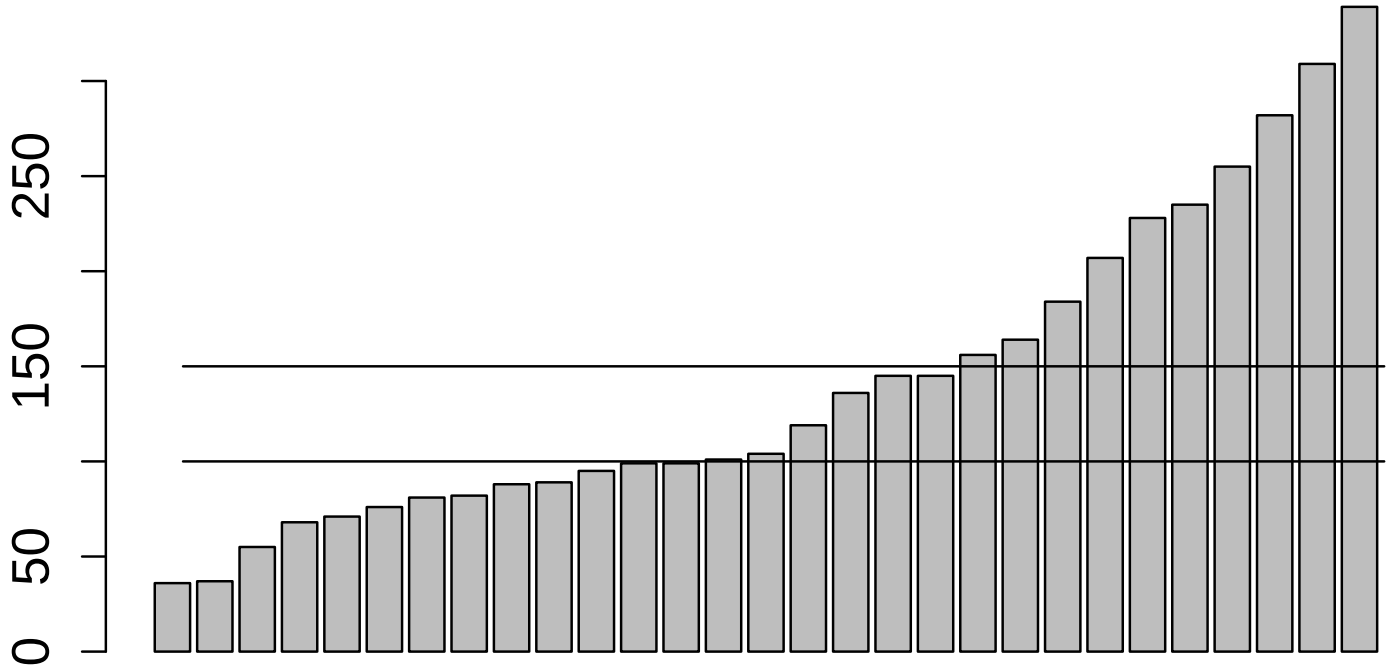
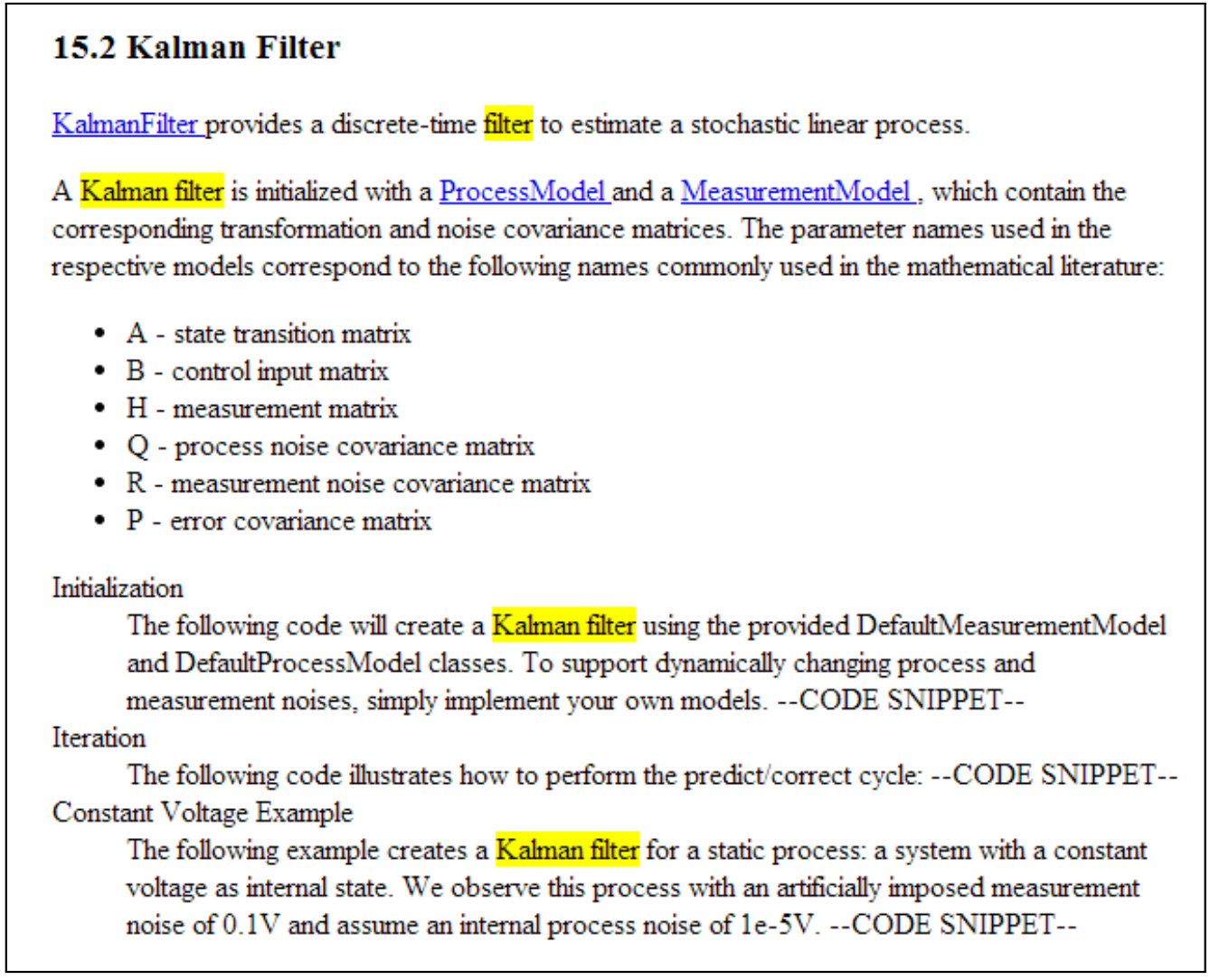


Figure : Section Lengths for JodaTime API Tutorial

Relevance Classification

- What is the data? - <API type, API section> pairs
- What are the categories? - relevant, irrelevant
- What classifier is used? - MaxEnt
- What are the features? - real-valued features, tutorial level, section level, sentence level, dependency features

Features - WordNum



15.2 Kalman Filter

KalmanFilter provides a discrete-time filter to estimate a stochastic linear process.

A Kalman filter is initialized with a ProcessModel and a MeasurementModel, which contain the corresponding transformation and noise covariance matrices. The parameter names used in the respective models correspond to the following names commonly used in the mathematical literature:

- A - state transition matrix
- B - control input matrix
- H - measurement matrix
- Q - process noise covariance matrix
- R - measurement noise covariance matrix
- P - error covariance matrix

Initialization

The following code will create a Kalman filter using the provided DefaultMeasurementModel and DefaultProcessModel classes. To support dynamically changing process and measurement noises, simply implement your own models. --CODE SNIPPET--

Iteration

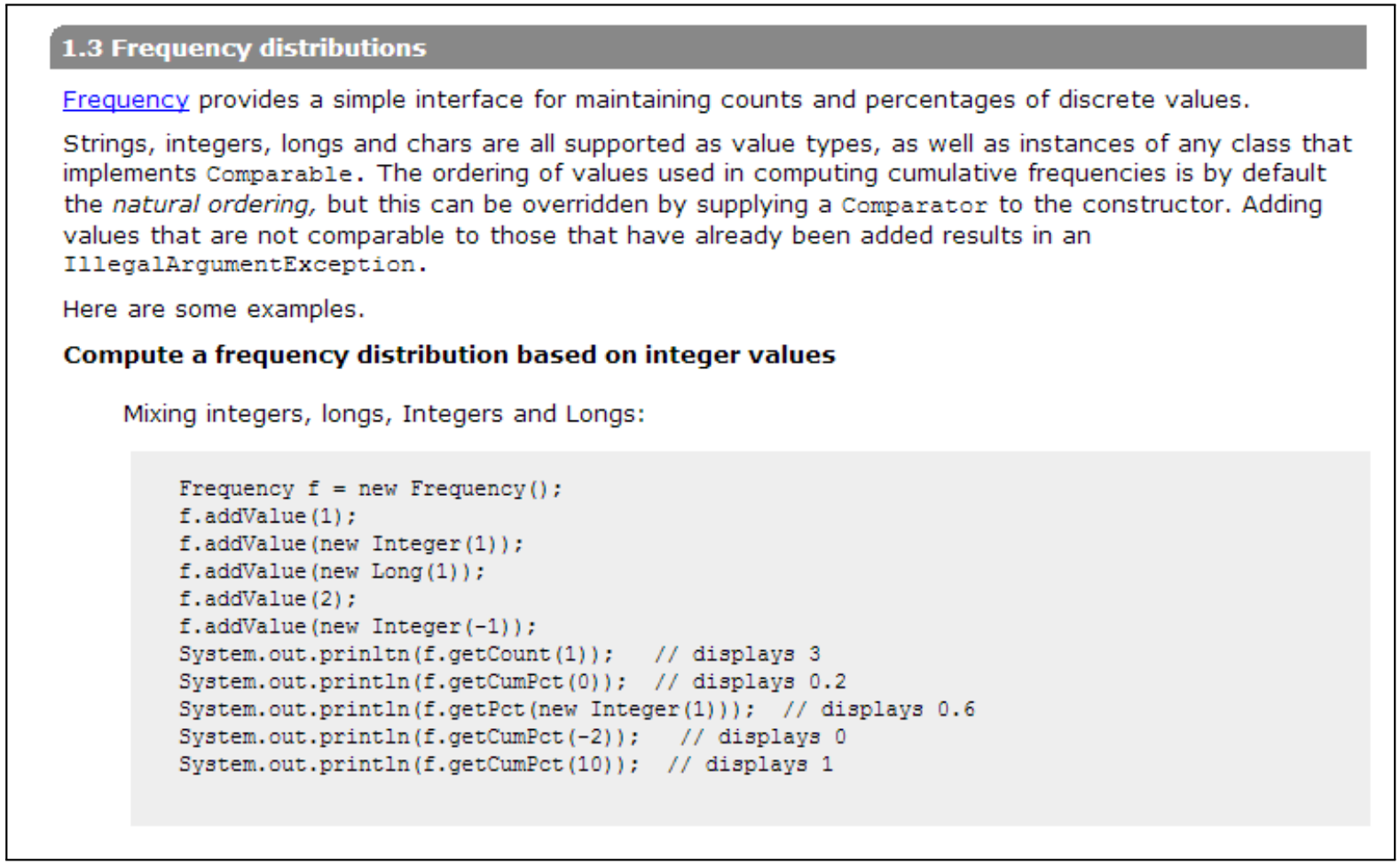
The following code illustrates how to perform the predict/correct cycle: --CODE SNIPPET--

Constant Voltage Example

The following example creates a Kalman filter for a static process: a system with a constant voltage as internal state. We observe this process with an artificially imposed measurement noise of 0.1V and assume an internal process noise of 1e-5V. --CODE SNIPPET--

Figure : A Section from Math Library Tutorial: Highlighted for `KalmanFilter` API Element

Features - inCode, notInCode



1.3 Frequency distributions

Frequency provides a simple interface for maintaining counts and percentages of discrete values.

Strings, integers, longs and chars are all supported as value types, as well as instances of any class that implements `Comparable`. The ordering of values used in computing cumulative frequencies is by default the *natural ordering,* but this can be overridden by supplying a `Comparator` to the constructor. Adding values that are not comparable to those that have already been added results in an `IllegalArgumentException`.

Here are some examples.

Compute a frequency distribution based on integer values

Mixing integers, longs, Integers and Longs:

```
Frequency f = new Frequency();
f.addValue(1);
f.addValue(new Integer(1));
f.addValue(new Long(1));
f.addValue(2);
f.addValue(new Integer(-1));
System.out.prinltn(f.getCount(1));   // displays 3
System.out.println(f.getCumPct(0));  // displays 0.2
System.out.println(f.getPct(new Integer(1)));  // displays 0.6
System.out.println(f.getCumPct(-2));   // displays 0
System.out.println(f.getCumPct(10));  // displays 1
```

Figure : A Section from Math Library Tutorial: inCode, notInCode feature example

Features - withCode

Compute summary statistics for a list of double values

Using the `DescriptiveStatistics` aggregate (values are stored in memory):

```
// Get a DescriptiveStatistics instance
DescriptiveStatistics stats = new DescriptiveStatistics();

// Add the data from the array
for( int i = 0; i < inputArray.length; i++) {
        stats.addValue(inputArray[i]);
}

// Compute some statistics
double mean = stats.getMean();
double std = stats.getStandardDeviation();
double median = stats.getPercentile(50);
```

Figure : A Section from Math Library Tutorial: withCode feature example

More Features

importantSent	"Alternatively, you can use the XMPPServer(ConnectionConfiguration) constructor to specify advanced connection settings"
enumeration	"In particular, we cover the usage of the key DateTime, Interval, Duration and Period classes."
example	"Each of these also has a corresponding property method, which returns a DateTime.Property binding to the appropriate field, such as year() or monthOfYear()"

A Few Examples of Dependencies

Example	Governor	Relation	Dependant	Total N	Positive N	Z-score	Norm.
... cannot use CLT...	use	dobjMDneg	clt	5	0	-2.85	0.32
... CLT specifies...	clt	nsubj	specify	11	11	2.6	0.93
... should catch CLT...	catch	dobjMD	clt	2	0	-1.81	0.44
... defined in CLT...	define	prepIN	clt	12	0	-4.42	0.19
... using CLT...	use	dobj	clt	88	61	1.42	0.79

API tutorials for Classification

API	Tutorial	Ref. name	N of words
JodaTime API	User guide	JodaTime	4659
apache.commons.math library	User guide	Math Library	28971
Java Collections Framework	Implementations in Java Tutorials	Collections(Official)	23583
Java Collections Framework	Tutorials by Jakob Jenkov	Collections(Jenkov)	12915
Smack API	Documentation	Smack	19075

- each tutorial annotated by two people
- disagreement discussion to reach a consensus

Evaluation

Leave One Out Cross Validation

- For each pairs of <API type, API section>
- Train classifier on the rest of the tutorial
- Test on that pair

For accumulated results calculate

Precision shows how noisy the retrieved items are

Recall shows how effective the algorithm is for finding relevant items

F1 harmonic mean of precision and recall

Table : LOOCV Results for All Tutorials

Tutorial	**P**	**R**	**F1**
JodaTime	0.81	0.73	0.77
Math Library	0.69	0.74	0.71
Collections(Official)	0.71	0.62	0.67
Collections(Jenkov)	0.84	0.76	0.80
Smack	0.87	0.80	0.83

Table : Cross Tutorial Testing Results

Test Tutorial	Precision	recall	F1
Jodatime	0.94	0.57	0.71
Math Library	0.87	0.48	0.62
Collections(Official)	0.74	0.76	0.75
Collections(Jenkov)	0.80	0.68	0.73
Smack	0.87	0.64	0.74

Table : Classification Results for Different Set of Features

Tutorial	JodaTime			Math Library			Collections (Official)			Collections (Jenkov)			Smack		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
RV	0.77	0.77	0.77	0.58	0.78	0.67	0.50	0.21	0.30	0.79	0.52	0.63	0.68	0.91	0.78
RV,T	0.84	0.87	0.85	0.56	0.65	0.60	0.62	0.23	0.34	0.84	0.62	0.71	0.73	0.73	0.73
RV,T,SC	0.82	0.77	0.79	0.62	0.74	0.68	0.66	0.48	0.56	0.83	0.81	0.82	0.85	0.79	0.81
RV,T,SC,ST	0.81	0.70	0.75	0.70	0.69	0.69	0.70	0.55	0.62	0.85	0.79	0.81	0.87	0.80	0.83
All	**0.88**	**0.77**	**0.82**	**0.69**	**0.74**	**0.71**	**0.71**	**0.62**	**0.67**	**0.84**	**0.76**	**0.80**	**0.87**	**0.80**	**0.83**

Figure : Percentage of wrongly classified cases of training and testing sets

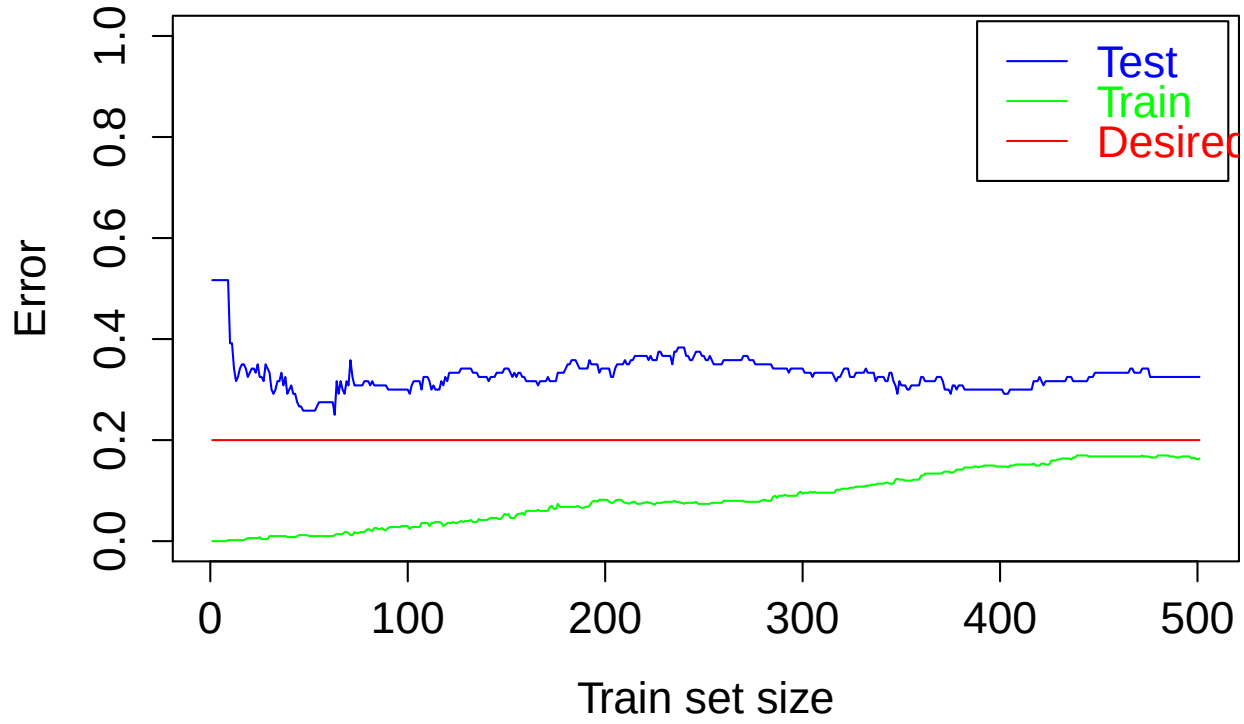


Figure : Precision for train and test sets

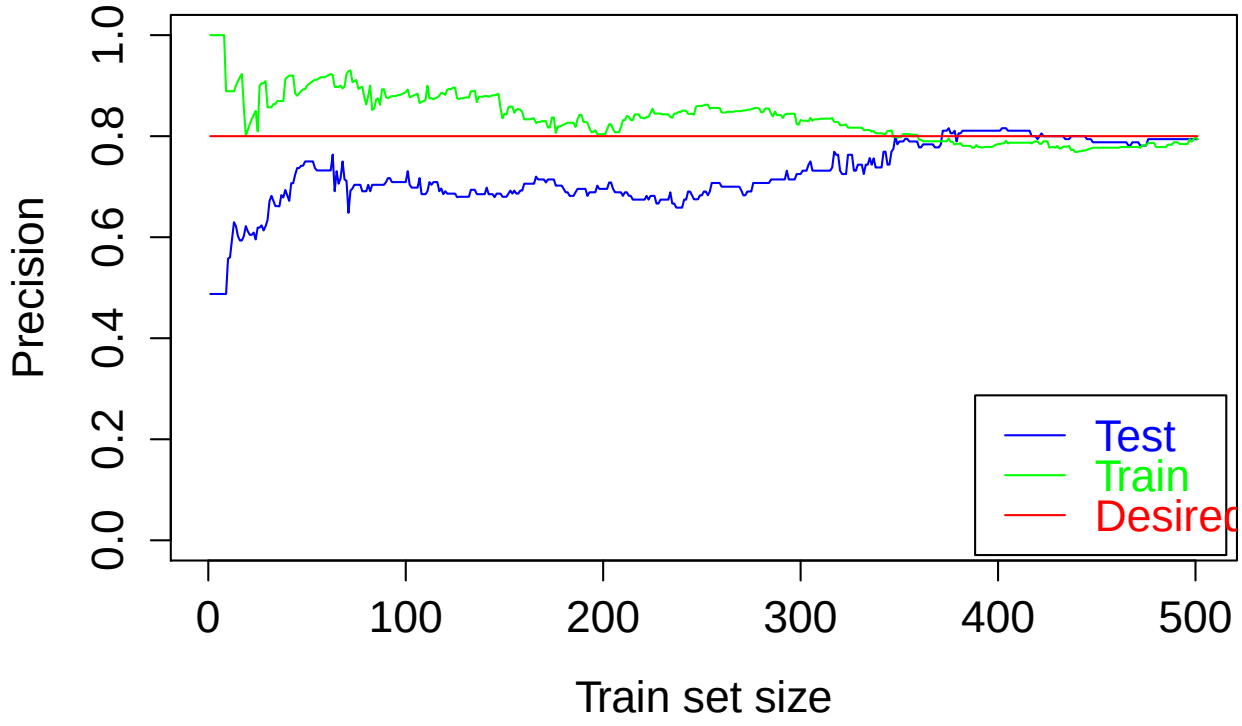


Figure : Recall for train and test sets