
Limdu.js

Limdu is a machine-learning framework for Node.js. It supports **multi-label classification**, **online learning**, and **real-time classification**. Therefore, it is especially suited for natural language understanding in dialog systems and chat-bots.

Limdu is in an “alpha” state - some parts are working (see this readme), but some parts are missing or not tested. Contributions are welcome.

Limdu currently runs on Node.js 0.12 and later versions.

Installation

```
1 npm install limdu
```

Demos

You can run the demos from this project: limdu-demo.

Table of Contents *generated with DocToc*

- Binary Classification
 - Batch Learning - learn from an array of input-output pairs:
 - Online Learning
 - Binding
 - Explanations
 - Other Binary Classifiers
- Multi-Label Classification
 - Other Multi-label classifiers
- Feature engineering
 - Feature extraction - converting an input sample into feature-value pairs:
 - Input Normalization
 - Feature lookup table - convert custom features to integer features
- Serialization
- Cross-validation
- Back-classification (aka Generation)
- SVM wrappers

-
- Undocumented features
 - Contributions
 - License

Binary Classification

Batch Learning - learn from an array of input-output pairs:

```
1 var limdu = require('limdu');
2
3 var colorClassifier = new limdu.classifiers.NeuralNetwork();
4
5 colorClassifier.trainBatch([
6   {input: { r: 0.03, g: 0.7, b: 0.5 }, output: 0}, // black
7   {input: { r: 0.16, g: 0.09, b: 0.2 }, output: 1}, // white
8   {input: { r: 0.5, g: 0.5, b: 1.0 }, output: 1}   // white
9 ]);
10
11 console.log(colorClassifier.classify({ r: 1, g: 0.4, b: 0 })); // 0.99
    - almost white
```

Credit: this example uses brain.js, by Heather Arthur.

Online Learning

```
1 var birdClassifier = new limdu.classifiers.Winnow({
2   default_positive_weight: 1,
3   default_negative_weight: 1,
4   threshold: 0
5 });
6
7 birdClassifier.trainOnline({'wings': 1, 'flight': 1, 'beak': 1, 'eagle':
8   : 1}, 1); // eagle is a bird (1)
9 birdClassifier.trainOnline({'wings': 0, 'flight': 0, 'beak': 0, 'dog':
10   : 1}, 0); // dog is not a bird (0)
11 console.dir(birdClassifier.classify({'wings': 1, 'flight': 0, 'beak':
12   : 0.5, 'penguin': 1})); // initially, penguin is mistakenly classified
    as 0 - "not a bird"
13 console.dir(birdClassifier.classify({'wings': 1, 'flight': 0, 'beak':
14   : 0.5, 'penguin': 1}, /*explanation level=*/4)); // why? because it
    does not fly.
15
16 birdClassifier.trainOnline({'wings': 1, 'flight': 0, 'beak': 1, '
17   penguin': 1}, 1); // learn that penguin is a bird, although it doesn't
    fly
```

```
13 birdClassifier.trainOnline({'wings': 0, 'flight': 1, 'beak': 0, 'bat': 1}, 0); // learn that bat is not a bird, although it does fly
14 console.dir(birdClassifier.classify({'wings': 1, 'flight': 0, 'beak': 1, 'chicken': 1})); // now, chicken is correctly classified as a bird, although it does not fly.
15 console.dir(birdClassifier.classify({'wings': 1, 'flight': 0, 'beak': 1, 'chicken': 1}, /*explanation level=*/4)); // why? because it has wings and beak.
```

Credit: this example uses Modified Balanced Margin Winnow (Carvalho and Cohen, 2006).

The “explanation” feature is explained below.

Binding

Using Javascript’s binding capabilities, it is possible to create custom classes, which are made of existing classes and pre-specified parameters:

```
1 var MyWinnow = limdu.classifiers.Winnow.bind(0, {
2     default_positive_weight: 1,
3     default_negative_weight: 1,
4     threshold: 0
5 });
6
7 var birdClassifier = new MyWinnow();
8 ...
9 // continue as above
```

Explanations

Some classifiers can return “explanations” - additional information that explains how the classification result has been derived:

```
1 var colorClassifier = new limdu.classifiers.Bayesian();
2
3 colorClassifier.trainBatch([
4     {input: { r: 0.03, g: 0.7, b: 0.5 }, output: 'black'},
5     {input: { r: 0.16, g: 0.09, b: 0.2 }, output: 'white'},
6     {input: { r: 0.5, g: 0.5, b: 1.0 }, output: 'white'},
7 ]);
8
9 console.log(colorClassifier.classify({ r: 1, g: 0.4, b: 0 },
10     /* explanation level = */1));
```

Credit: this example uses code from classifier.js, by Heather Arthur.

The explanation feature is experimental and is supported differently for different classifiers. For example, for the Bayesian classifier it returns the probabilities for each category:

```
1 { classes: 'white',  
2   explanation: [ 'white: 0.0621402182289608', 'black:  
0.031460948468170505' ] }
```

While for the winnow classifier it returns the relevance (feature-value times feature-weight) for each feature:

```
1 { classification: 1,  
2   explanation: [ 'bias+1.12', 'r+1.08', 'g+0.25', 'b+0.00' ] }
```

WARNING: The internal format of the explanations might change without notice. The explanations should be used for presentation purposes only (and not, for example, for extracting the actual numbers).

Other Binary Classifiers

In addition to Winnow and NeuralNetwork, version 0.2 includes the following binary classifiers:

- Bayesian - uses classifier.js, by Heather Arthur.
- Perceptron - Loosely based on perceptron.js, by John Chesley.
- SVM - uses svm.js, by Andrej Karpathy.
- Linear SVM - wrappers around SVM-Perf and Lib-Linear (see below).
- Decision Tree - based on node-decision-tree-id3 by Ankit Kuwadekar or ID3-Decision-Tree by Will Kurt.

This library is still under construction, and not all features work for all classifiers. For a full list of the features that do work, see the “test” folder.

Multi-Label Classification

In binary classification, the output is 0 or 1;

In multi-label classification, the output is a set of zero or more labels.

```
1 var MyWinnow = limdu.classifiers.Winnow.bind(0, {retrain_count: 10});  
2  
3 var intentClassifier = new limdu.classifiers.multilabel.BinaryRelevance  
4   ({  
5     binaryClassifierType: MyWinnow  
6   });
```

```
7 intentClassifier.trainBatch([
8   {input: {I:1,want:1,an:1,apple:1}, output: "APPLE"},
9   {input: {I:1,want:1,a:1,banana:1}, output: "BANANA"},
10  {input: {I:1,want:1,chips:1}, output: "CHIPS"}
11  ]);
12
13 console.dir(intentClassifier.classify({I:1,want:1,an:1,apple:1,and:1,a
    :1,banana:1})); // ['APPLE', 'BANANA']
```

Other Multi-label classifiers

In addition to BinaryRelevance, version 0.2 includes the following multi-label classifier types (see the multilabel folder):

- Cross-Lingual Language Model Classifier (based on Anton Leusky and David Traum, 2008)
- HOMER - Hierarchy Of Multi-label classiFIErs (based on Tsoumakas et al., 2007)
- Meta-Labeler (based on Lei Tang, Suju Rajan, Vijay K. Narayanan, 2009)
- Joint identification and segmentation (based on Fabrizio Morbini, Kenji Sagae, 2011)
- Passive-Aggressive (based on Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, Yoram Singer, 2006)
- Threshold Classifier (converting multi-class classifier to multi-label classifier by finding the best appropriate threshold)

This library is still under construction, and not all features work for all classifiers. For a full list of the features that do work, see the “test” folder.

Feature engineering

Feature extraction - converting an input sample into feature-value pairs:

```
1 // First, define our base classifier type (a multi-label classifier
   based on winnow):
2 var TextClassifier = limdu.classifiers.multilabel.BinaryRelevance.bind
   (0, {
3   binaryClassifierType: limdu.classifiers.Winnow.bind(0, {
4     retrain_count: 10})
5 });
6 // Now define our feature extractor - a function that takes a sample
   and adds features to a given features set:
7 var WordExtractor = function(input, features) {
8   input.split(" ").forEach(function(word) {
9     features[word]=1;
```

```

10     });
11 };
12
13 // Initialize a classifier with the base classifier type and the
14   feature extractor:
15 var intentClassifier = new limdu.classifiers.EnhancedClassifier({
16     classifierType: TextClassifier,
17     featureExtractor: WordExtractor
18 });
19
20 // Train and test:
21 intentClassifier.trainBatch([
22     {input: "I want an apple", output: "apl"},
23     {input: "I want a banana", output: "bnn"},
24     {input: "I want chips", output: "cps"},
25 ]);
26 console.dir(intentClassifier.classify("I want an apple and a banana"));
27 // ['apl','bnn']
28 console.dir(intentClassifier.classify("I WANT AN APPLE AND A BANANA"));
29 // []

```

As you can see from the last example, by default feature extraction is case-sensitive. We will take care of this in the next example.

Instead of defining your own feature extractor, you can use those already bundled with limdu:

```

1 limdu.features.NGramsOfWords
2 limdu.features.NGramsOfLetters
3 limdu.features.HypernymExtractor

```

You can also make 'featureExtractor' an array of several feature extractors, that will be executed in the order you include them.

Input Normalization

```

1 //Initialize a classifier with a feature extractor and a case
2   normalizer:
3 intentClassifier = new limdu.classifiers.EnhancedClassifier({
4     classifierType: TextClassifier, // same as in previous example
5     normalizer: limdu.features.LowerCaseNormalizer,
6     featureExtractor: WordExtractor // same as in previous example
7 });
8
9 //Train and test:
10 intentClassifier.trainBatch([
11     {input: "I want an apple", output: "apl"},
12     {input: "I want a banana", output: "bnn"},
13     {input: "I want chips", output: "cps"},

```

```
13     });
14
15     console.dir(intentClassifier.classify("I want an apple and a banana"));
16     // ['apl', 'bnn']
17
18     console.dir(intentClassifier.classify("I WANT AN APPLE AND A BANANA"));
19     // ['apl', 'bnn']
```

Of course you can use any other function as an input normalizer. For example, if you know how to write a spell-checker, you can create a normalizer that corrects typos in the input.

You can also make ‘normalizer’ an array of several normalizers. These will be executed in the order you include them.

Feature lookup table - convert custom features to integer features

This example uses the quadratic SVM implementation `svm.js`, by Andrej Karpathy. This SVM (like most SVM implementations) works with integer features, so we need a way to convert our string-based features to integers.

```
1 var limdu = require('limdu');
2
3 // First, define our base classifier type (a multi-label classifier
4 // based on svm.js):
5 var TextClassifier = limdu.classifiers.multilabel.BinaryRelevance.bind(
6   0, {
7     binaryClassifierType: limdu.classifiers.SvmJs.bind(0, {C: 1.0})
8   });
9
10 // Initialize a classifier with a feature extractor and a lookup table:
11 var intentClassifier = new limdu.classifiers.EnhancedClassifier({
12   classifierType: TextClassifier,
13   featureExtractor: limdu.features.NGramsOfWords(1), // each word
14   // ("1-gram") is a feature
15   featureLookupTable: new limdu.features.FeatureLookupTable()
16 });
17
18 // Train and test:
19 intentClassifier.trainBatch([
20   {input: "I want an apple", output: "apl"},
21   {input: "I want a banana", output: "bnn"},
22   {input: "I want chips", output: "cps"},
23 ]);
24
25 console.dir(intentClassifier.classify("I want an apple and a banana"));
26 // ['apl', 'bnn']
```

The `FeatureLookupTable` takes care of the numbers, while you may continue to work with texts!

Serialization

Say you want to train a classifier on your home computer, and use it on a remote server. To do this, you should somehow convert the trained classifier to a string, send the string to the remote server, and deserialize it there.

You can do this with the “serialization.js” package:

```
1 npm install serialization
```

On your home machine, do the following:

```
1 var serialize = require('serialization');
2
3 // First, define a function that creates a fresh (untrained)
  classifier.
4 // This code should be stand-alone - it should include all the 'require
  ' statements
5 //   required for creating the classifier.
6 function newClassifierFunction() {
7     var limdu = require('limdu');
8     var TextClassifier = limdu.classifiers.multilabel.BinaryRelevance.
        bind(0, {
9         binaryClassifierType: limdu.classifiers.Winnow.bind(0, {
            retrain_count: 10})
10    });
11
12    var WordExtractor = function(input, features) {
13        input.split(" ").forEach(function(word) {
14            features[word]=1;
15        });
16    };
17
18    // Initialize a classifier with a feature extractor:
19    return new limdu.classifiers.EnhancedClassifier({
20        classifierType: TextClassifier,
21        featureExtractor: WordExtractor,
22        pastTrainingSamples: [], // to enable retraining
23    });
24 }
25
26 // Use the above function for creating a new classifier:
27 var intentClassifier = newClassifierFunction();
28
29 // Train and test:
30 var dataset = [
31     {input: "I want an apple", output: "apl"},
32     {input: "I want a banana", output: "bnn"},
33     {input: "I want chips", output: "cps"},
34 ];
```

```

35 intentClassifier.trainBatch(dataset);
36
37 console.log("Original classifier:");
38 intentClassifier.classifyAndLog("I want an apple and a banana"); // ['
    apl', 'bnn']
39 intentClassifier.trainOnline("I want a doughnut", "dnt");
40 intentClassifier.classifyAndLog("I want chips and a doughnut"); // ['
    cps', 'dnt']
41 intentClassifier.retrain();
42 intentClassifier.classifyAndLog("I want an apple and a banana"); // ['
    apl', 'bnn']
43 intentClassifier.classifyAndLog("I want chips and a doughnut"); // ['
    cps', 'dnt']
44
45 // Serialize the classifier (convert it to a string)
46 var intentClassifierString = serialize.toString(intentClassifier,
    newClassifierFunction);
47
48 // Save the string to a file, and send it to a remote server.

```

On the remote server, do the following:

```

1 // retrieve the string from a file and then:
2
3 var intentClassifierCopy = serialize.fromString(intentClassifierString,
    __dirname);
4
5 console.log("Deserialized classifier:");
6 intentClassifierCopy.classifyAndLog("I want an apple and a banana");
    // ['apl', 'bnn']
7 intentClassifierCopy.classifyAndLog("I want chips and a doughnut"); //
    ['cps', 'dnt']
8 intentClassifierCopy.trainOnline("I want an elm tree", "elm");
9 intentClassifierCopy.classifyAndLog("I want doughnut and elm tree");
    // ['dnt', 'elm']

```

CAUTION: Serialization was not tested for all possible combinations of classifiers and enhancements. Test well before use!

Cross-validation

```

1 // create a dataset with a lot of input-output pairs:
2 var dataset = [ ... ];
3
4 // Decide how many folds you want in your k-fold cross-validation:
5 var numOfFolds = 5;
6
7 // Define the type of classifier that you want to test:
8 var IntentClassifier = limdu.classifiers.EnhancedClassifier.bind(0, {

```

```

 9     classifierType: limdu.classifiers.multilabel.BinaryRelevance.bind
10         (0, {
11         binaryClassifierType: limdu.classifiers.Winnow.bind(0, {
12             retrain_count: 10})
13     });
14
15     var microAverage = new limdu.utils.PrecisionRecall();
16     var macroAverage = new limdu.utils.PrecisionRecall();
17
18     limdu.utils.partitions.partitions(dataset, numOfFolds, function(
19         trainSet, testSet) {
20         console.log("Training on "+trainSet.length+" samples, testing on "+
21             testSet.length+" samples");
22         var classifier = new IntentClassifier();
23         classifier.trainBatch(trainSet);
24         limdu.utils.test(classifier, testSet, /* verbosity = */0,
25             microAverage, macroAverage);
26     });
27
28     macroAverage.calculateMacroAverageStats(numOfFolds);
29     console.log("\n\nMACRO AVERAGE:"); console.dir(macroAverage.fullStats()
30         );
31
32     microAverage.calculateStats();
33     console.log("\n\nMICRO AVERAGE:"); console.dir(microAverage.fullStats()
34         );

```

Back-classification (aka Generation)

Use this option to get the list of all samples with a given class.

```

1  var intentClassifier = new limdu.classifiers.EnhancedClassifier({
2      classifierType: limdu.classifiers.multilabel.BinaryRelevance.bind
3          (0, {
4          binaryClassifierType: limdu.classifiers.Winnow.bind(0, {
5              retrain_count: 10})
6      });
7
8      featureExtractor: limdu.features.NGramsOfWords(1),
9      pastTrainingSamples: [],
10  });
11
12  // Train and test:
13  intentClassifier.trainBatch([
14      {input: "I want an apple", output: "apl"},
15      {input: "I want a banana", output: "bnn"},
16      {input: "I really want an apple", output: "apl"},
17      {input: "I want a banana very much", output: "bnn"},

```

```
15     });
16
17     console.dir(intentClassifier.backClassify("apl")); // [ 'I want an
    apple', 'I really want an apple' ]
```

SVM wrappers

The native svm.js implementation takes a lot of time to train - quadratic in the number of training samples. There are two common packages that can be trained in time linear in the number of training samples. They are:

- SVM-Perf - by Thorsten Joachims;
- LibLinear - Fan, Chang, Hsieh, Wang and Lin.

The limdu.js package provides wrappers for these implementations. In order to use the wrappers, you must have the binary file used for training in your path, that is:

- **svm_perf_learn** - from SVM-Perf.
- **liblinear_train** - from LibLinear.

Once you have any one of these installed, you can use the corresponding classifier instead of any binary classifier used in the previous demos, as long as you have a feature-lookup-table. For example, with SvmPerf:

```
1 var intentClassifier = new limdu.classifiers.EnhancedClassifier({
2   classifierType: limdu.classifiers.multilabel.BinaryRelevance.bind
    (0, {
3     binaryClassifierType: limdu.classifiers.SvmPerf.bind(0, {
4       learn_args: "-c 20.0"
5     })
6   }),
7   featureExtractor: limdu.features.NGramsOfWords(1),
8   featureLookupTable: new limdu.features.FeatureLookupTable()
9 });
```

and similarly with SvmLinear.

See the files classifiers/svm/SvmPerf.js and classifiers/svm/SvmLinear.js for a documentation of the options.

Undocumented features

Some advanced features are working but not documented yet. If you need any of them, open an issue and I will try to document them.

-
- Custom input normalization, based on regular expressions.
 - Input segmentation for multi-label classification - both manual (with regular expressions) and automatic.
 - Feature extraction for model adaptation.
 - Spell-checker features.
 - Hypernym features.
 - Classification based on a cross-lingual language model.
 - Format conversion - ARFF, JSON, svm-light, TSV.

License

LGPL

Contributions

Code contributions are welcome. Reasonable pull requests, with appropriate documentation and unit-tests, will be accepted.

Do you like limdu? Remember that you can star it :-)