Kernel Barcoding Library v1.0
Analysis and Prediction for Barcode data

March 9, 2009

Feel free to use this software in your research. If you do use it, please cite the following papers:

```
@inproceedings{nips2008inexact,
  author = {Pavel Kuksa and Pai-Hsi Huang and Vladimir Pavlovic},
  title = {Scalable Algorithms for String Kernels with Inexact Matching},
  booktitle = {NIPS},
  year = {2008},
  pages = {}
}
@inproceedings{icpr2008sssk,
  author = {Pavel Kuksa and Pai-Hsi Huang and Vladimir Pavlovic},
  title =  {Fast Protein Homology and Fold Detection with
           Sparse Spatial Sample Kernels},
  booktitle = {ICPR},
  year = {2008},
  pages = {}
}
@inproceedings{wabi2007barcoding,
  author    = {Pavel Kuksa and
               Vladimir Pavlovic},
  title     = {Fast Kernel Methods for {SVM} Sequence Classifiers},
  booktitle = {WABI},
  year      = {2007},
  pages     = {228-239},
  ee        = {http://dx.doi.org/10.1007/978-3-540-74126-8_22},
}
```

This implements new kernel algorithms for strings (see references).

1 Installation

Copy the tar file kernelbarcode-0.1c.tar.gz to your directory

```
> tar xzvf kernelbarcode-1.0c.tar.gz
> cd kernelbarcode-1.0c
> make
> make install
```

This will produce object files and executables for string kernel computations (in bin directory):
mismatchKernel : computes mismatch kernel matrices
gappedKernel : computes gapped kernel matrices
wildcardKernel : computes wildcard kernel matrices
tripleKernel : computes triple kernel matrices
doubleKernel : computes double kernel matrices

Requirements:
You will need LibSVM library (`www.csie.ntu.edu.tw/~cjlin/libsvm/`), Python, and gcc compiler.

IMPORTANT: Note that for SVM/LibSVM to work "libsvmpath" has to be set to a correct path to LibSVM directory, e.g. `/your_directory/libsvm/` You can set path by setting "libsvmpath" variable in the main python script "kernel-barcoding.py"

2 Usage

Main program file is "kernel-barcoding.py" which is a Python script around C codes that implement state-of-the-art string algorithms for computing barcode similarities.

Main tasks that can be performed are

1. computing barcode similarities between barcodes in a given barcode collection ("kernel" mode), e.g.

   ```
   python kernel-barcoding.py -t kernel -K double datasets/ast.fasta
   ```

2. computing barcode features for a given collection of barcodes ("features" mode), e.g.

   ```
   python kernel-barcoding.py -t feature -K double datasets/ast.fasta
   ```

3. computing n-fold cross-validation classification error estimates for a given barcode collection, e.g. SVM classifier (5-fold):

   ```
   python kernel-barcoding.py -t svm -K double -V 5 datasets/ast.fasta
   ```

   Nearest-neighbor classifier (5-fold):

```
python kernel-barcoding.py -t nn -K double -V 5 datasets/ast.fasta
```

4. Predicting assignments of queries using reference barcodes ("train" data), e.g.

   ```
   python kernel-barcoding.py -t predict -K double -Q datasets/ast-test.fasta
    datasets/ast-train.fasta
   ```

   IMPORTANT: LibSVM is required with SVM-based tasks (i.e. "svm", "predict" modes)

Directory "datasets" contains several benchmark barcode datasets (see description below)

Directory `reference_outputs` contains outputs generated by running "predict" task on Astraptes dataset. You can reproduce this by running

```
python kernel-barcoding.py -t predict -K -triple -d 3 -Q datasets/ast-test.fasta
 datasets/ast-train.fasta
```

2.1 Input formats

Input files with barcode sequences should be in `fasta` format with headers consisting of three fields (third field is the barcode class (species name), see fasta files under datasets folder).

2.2 Main program options

```
-t : Task type. Can be
      kernel
      feature
      svm
      nn
      predict
-K : Kernel type. Can be
      double
      triple
      mismatch
      wildcard
      gapped
-Q : Query file. Used with "predict" task.
-V : Number of folds for cross-validation.
-d : max. distance between samples for spatial kernels (double/triple)
-k : k-mer length (for mismatch, gapped, wildcard kernels)
-w : max. number of wildcards (parameter of the wildcard kernels)
-m : max. number of mismatches (parameter of the mismatch kernel)
```

Any of the tasks can be combined with any of the kernel types. NOTE: As of now, for the "feature" task only double/triple features are supported.

3 Tasks

3.1 Evaluating prediction performance using cross-validation

To evaluate classifier performance for a given barcode dataset, you can use "svm" and "nn" modes to compute n-fold cross-validation error estimates for SVM or Nearest-neighbor classifiers. E.g. 5-fold crossvalidation on Astraptes dataset:

```
python kernel-barcoding.py -t svm -K double -V 5 datasets/ast.fasta
python kernel-barcoding.py -t nn -K double -V 5 datasets/ast.fasta
```

IMPORTANT: Note that for SVM to work "libsvmpath" has to be set to a correct path to LibSVM directory, e.g. /usr/bin/libsvm/. You can set path by setting "libsvmpath" variable in the main python script "kernel-barcoding.py"

3.2 Computing assignments of queries

Using "predict" mode, you can make assignments for query barcode sequences using "train" barcode set with known classification.

```
python kernel-barcoding.py -t predict -K double -Q datasets/ast-test.fasta
datasets/ast-train.fasta
```

This will produce predictions for query sequences in "ast-test.fasta" file using train sequences in "ast-train.fasta" as a reference, i.e. train data. The output will consist of the following files:
"predictions.txt" - lists predicted class assignment
"ranking.txt" - lists closest match (i.e. nearest neighbor) for each query sequence along with confidence score ("distance" to the nearest neighbor).
"model.libsvm" - trained SVM model in LibSVM format.
"kernel.train", "kernel.test", "kernel.query" - similarity/kernel matrices of size $|train| \times |train||$, $|test| \times |train|$, and $|test| \times |test|$, respectively.

3.3 Computing barcode similarity

If you want to evaluate similarity between barcodes in a given collection of barcodes, you can use the "kernel" mode, e.g.

```
python kernel-barcoding.py -t kernel -K double datasets/ast.fasta
```

This will produce files "kernel.txt" and "kernel.txt.normalized" with similarity scores between every pair of barcodes stored in a square matrix ("kernel.txt.normalized" will contain normalized similarity scores).

3.4 Computing barcode features

If you want compute barcode features, you can do it by using "features" mode, e.g.

```
python kernel-barcoding.py -t features -K double datasets/ast.fasta
```

This will produce file with barcode features ”features.txt” with corresponding feature list in ”featuremap.txt”.

3.5 Outputs

Depending on your task (e.g. computing features, or barcode similarity), system will produce different output files. We describe them below:

classnames.txt - names (string) of classes; this is generated based on the ”train” file

datalabels.txt - labels (integers) for barcode sequences; this is based on classnames.txt

cv_idx.txt - crossvalidation indices (integers), for each sequence indicates fold number in which the sequence used as test.

kernel.train, kernel.test, kernel.query - similarity/kernel matrices of size $|train| \times |train||$, $|test| \times |train|$, and $|test| \times |test|$, respectively. These are produced in ”predict” mode only (i.e. using train and test files).

kernel.txt - similarity/kernel matrix. Will be produced in ”kernel”, ”svm”, or ”nn” modes. This is a square matrix containing pairwise similarities between each pair of barcode sequences.

predictions.txt - contains query assignments <class(integer), class_name(string)>

ranking.txt - contains closest match from train file for each of the query sequence

```
<query\_sequence_header> <closest_match_sequence_header> <confidence>
```

4 Datasets

datasets directory contains several benchmark barcode collections in fasta file formats:
ast.fasta - Astraptes (12 groups)
hesperiidae.fasta - Hesperiidae species
acg.fasta - Hesperiidae of the ACG 1
batsGuyana.fasta - Bats of Guyana
birds2.fasta - Birds of North America - Phase II

For details on these datasets see Barcode Research Challenges page
`http://dimacs.rutgers.edu/Workshops/BarcodeResearchChallenges2007/`
All of the string kernels functions take a text file with sequences and output a text file with a Gram matrix.

5 Using with libsvm

Kernel computation utilities in bin directory can be used on their own to compute similarity/kernel matrices (see details below and help for particular function) in text format. clibsvm can be used then to convert text file with Gram matrix to libsvm format. That can be used with libsvm -t 4 option for training/testing. clibsvm usage:

```
./clibsvm <kernel-file> <label-file>
```

`<label-file>` is a text file with one label per line, labels assumed to be integers, i.e. line `<i>` contains integer label for example `<i>`.

E.g. bin/doubleKernel datasets/ast.integer 5 4
produces Gram kernel matrix in kernel.txt
bin/clibsvm kernel.txt datasets/ast.lab
produces kernel.txt.libsvm
svm-train -t 4 -c 10 -v 10 kernel.txt.libsvm
runs 10-fold cross-validation using precomputed kernel.

String kernels are called with the following parameters:

```
mismatchKernel <Sequence-file> <k> <m> <Alphabet-size>
gappedKernel <Sequence-file> <k> <g> <Alphabet-size>
wildcardKernel <Sequence-file> <k> <w> <Alphabet-size>
doubleKernel <Sequence-file> <d> <Alphabet-size>
tripleKernel <Sequence-file> <d> <Alphabet-size>

where
<Sequence-file> is the file with sequence data:
one sequence per line (line should end with line feed),
with sequence elements separated by space,
all sequence elements are assumed to be in the range
[0, <AlphabetSize> - 1]. See datasets/ast.integer for an example of the
sequence file format.
<k>,<m>,<g>,<w>,<d> are corresponding kernel parameters (see references
and help for a particular function for details)
<Alphabet-size> is the size of the alphabet (e.g. for DNA,
<Alphabet-size> is 4)
```

Output kernel matrix is written into
kernel.txt
file.

6 ToDo

- This is a preliminary version for research purposes. Parameters, option handling, checking need to be added.

7 Authors

Pavel Kuksa
pkuksa@cs.rutgers.edu
Vladimir Pavlovic
vladimir@cs.rutgers.edu