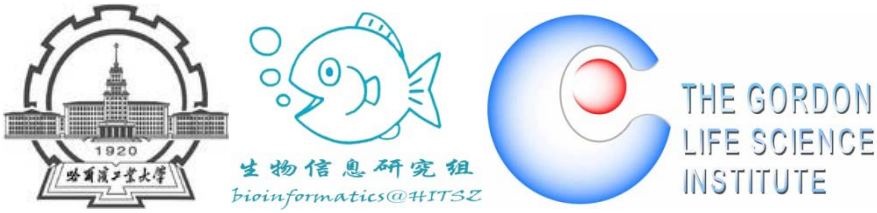
repDNA: a Python package to generate various modes of feature vectors for DNA sequences by incorporating user-defined physicochemical properties and sequence-order effects

Package Version: Release 1.1.1

2014-11-04



Contents

1. Introduction.....2
2. Application in bioinformatics.....2
2.1 Predicting DNaseI HSs in the human genome with reverse compliment kmers.....3
2.2 Predicting nucleosome positioning in genomes with dinucleotide-based auto covariance.....6
2.3 Identifying recombination spots with pseudo dinucleotide composition.....9
Reference 12

1. Introduction

The repDNA Python package can generate various feature vectors for DNA sequences, this Python package could:

1) Calculate three nucleic acid composition features describing the local sequence information by means of kmers (subsequences of DNA sequences);

2) Calculate six autocorrelation features describing the level of correlation between two oligonucleotides along a DNA sequence in terms of their specific physicochemical properties;

3) Calculate six pseudo nucleotide composition features, which can be used to represent a DNA sequence with a discrete model or vector yet still keep considerable sequence order information, particularly the global or long-range sequence order information, via the physicochemical properties of its constituent oligonucleotides.

There are four modules in the repDNA package, including util, nac, ac and psenac. The util module contains several basic functions manipulating DNA data, including reading DNA data from files or list (a data structure in Python), checking the validity and normalizing the user-defined physicochemical indices, etc. The three modules nac, ac and psenac respond to the calculation of the 15 different features from three feature categories. In order to use the repDNA package to calculate these features as needed, the users need to import the appropriate class from the corresponding module, construct a responding object, and then call the corresponding methods to calculate these features.

The repDNA package is available from http://bioinformatics.hitsz.edu.cn/repDNA/.

If the user has installed **pip** (a tool for installing and managing Python packages), just type:

```
pip install repDNA
```

Otherwise, download and unzip the repDNA package, go to the directory, and type:

```
python setup.py install
```

or

```
sudo python setup.py install
```

2. Application in bioinformatics

The repDNA would be applied to solve many tasks in the field of bioinformatics. We will introduce three examples of its applications in computational genomics, including DNaseI HS prediction, nucleosome positioning prediction, and recombination spot

identification. All datasets and Python scripts used in the next three examples can be download on http://bioinformatics.hitsz.edu.cn/repDNA/download.

2.1 Predicting DNaseI HSs in the human genome with reverse compliment kmers

In the living cell nucleus, genomic DNA is packaged into chromatin. DNA sequences that regulate transcription and other chromosomal processes are associated with local disruptions, or 'openings', in chromatin structure caused by the cooperative action of regulatory proteins (Noble, et al., 2005). In most cases, identification of functional elements marked by HSs significantly preceded the assignment of a specific functional role (enhancer, insulator, etc.) to those elements (Gross and Garrard, 1998) (Li, et al., 2002). Here we re-built the experiments reported in (Noble, et al., 2005) to show how to use the repDNA Python package and some third-party Python packages to construct a predictor for predicting DNaseI HSs in the human genome.

In order to construct a computational predictor to identify DNaseI HSs in the human genome, the reverse compliment kmers were extracted as features to capture the characteristics of DNA sequences, and the SVMs were employed as the classifiers following the experiments reported in the paper (Noble, et al., 2005). The 10-fold cross-validation was used to evaluate the performance of the predictor. 280 validated erythroid HS sequences from throughout the human genome were treated as the positive sample set, which were stored in a file named "hs.fasta". 737 sequences from around the genome (distributed proportionally among the autosomes and X chromosome but excepting the Y chromosome) that were non-hypersensitive when tested in the same cell type were treated as negative sample set stored in a file named "non-hs.fasta". These two files can be found in folder "repDNA/repDNA/example/", and the source code of this experiment can be found in a Python script file "example1.py", located in "repDNA/repDNA/example/example1.py". Of course, the user can also easily find this files in repDNA home-page download page. In order to use the SVM classifiers and plot the ROC curves, we employed four third-part Python packages, including numpy(Walt, et al., 2011), sklearn(Pedregosa, et al., 2011), scipy(Jones E, 2001), matplotlib(Hunter, 2007).

First, we should import the RevcKmer class from repDNA.nac module to construct a RevcKmer object, and call make_revckmer_vec method with parameters *k*=6, *normalize*=True, *upto*=True (the optimized values of the two parameters as reported in (Noble, et al., 2005)) to generate the corresponding feature vectors of DNA sequences in the dataset:

```
from repDNA.nac import RevcKmer

# Generate the feature vectors based on reverse compliment kmer.
rev_kmer = RevcKmer(k=6, normalize=True, upto=True)
pos_vec = rev_kmer.make_revckmer_vec(open('hs.fasta'))
```

```
neg_vec = rev_kmer.make_revckmer_vec(open('non-hs.fasta'))
```

The feature vectors of positive and negative samples were stored in two lists pos_vec and neg_vec, respectively. If the format of the input files is not correct, the corresponding error messages will be shown on the screen, and the program will be terminated.

```
print(len(pos_vec))
280
print(len(neg_vec))
737
```

Next, we merge the positive and negative feature vectors into a numpy.array (a data type in numpy package), and generate the corresponding labels to represent their classes (1 represents the positive samples, and 0 represents the negative samples).

```
import numpy as np

# Merge positive and negative feature vectors and generate their
# corresponding labels.
vec = np.array(pos_vec + neg_vec)
vec_label = np.array([1] * len(pos_vec) + [0] * len(neg_vec))
```

Now, the SVMs in sklearn package were employed as the classifiers with linear kernel function. The performance of this predictor was evaluated by using 10-fold cross-validation, and its accuracy will be calculated and shown on the screen.

```
from sklearn import svm
from sklearn import cross_validation

# Using 10-fold cross-validation to evaluate the performance of the
# predictor.
clf = svm.LinearSVC()
scores= cross_validation.cross_val_score(clf, vec, y=vec_label, cv=10)
print('Per accuracy in 10-fold CV:')
print(scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std()*2))
```

```
Per accuracy in 10-fold CV:
[0.79411765  0.87254902  0.81372549  0.85294118  0.87254902  0.8627451
 0.80392157  0.84158416  0.84158416  0.85148515]
Accuracy: 0.84 (+/- 0.05)
```

Besides, we can also plot the mean ROC curve by using sklearn, scipy and matplotlib packages:

```
from sklearn.cross_validation import StratifiedKFold
from sklearn.metrics import roc_curve, auc
from scipy import interp
import matplotlib.pyplot as plt

# evaluate the predictor by using 10-fold cross-validation and
# plot the mean ROC curve.
cv = StratifiedKFold(vec_label, n_folds=10)
classifier = svm.SVC(kernel='linear', probability=True)

mean_tpr = 0.0
mean_fpr = np.linspace(0, 1, 100)
all_tpr = []

for i, (train, test) in enumerate(cv):
    probas_=classifier.fit(vec[train],
                           vec_label[train]).predict_proba(vec[test])
    # Compute ROC curve and AUC.
    fpr, tpr, thresholds = roc_curve(vec_label[test], probas_[:, 1])
    mean_tpr += interp(mean_fpr, fpr, tpr)
    mean_tpr[0] = 0.0

# Plot the ROC curve.
mean_tpr /= len(cv)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
plt.plot(mean_fpr, mean_tpr, '-',
          label='Mean ROC (area = %0.2f)' % mean_auc, lw=2)

plt.xlim([0, 1.0])
plt.ylim([0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.show()
```

The mean ROC curve of the predictor was shown in **Fig. 1**.

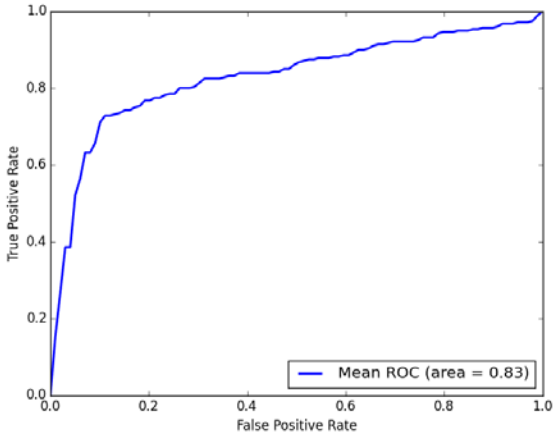


Figure 1. ROC curve achieved by the predictor based on reverse compliment kmers for predicting DNaseI HSs in the human genome.

2.2 Predicting nucleosome positioning in genomes with dinucleotide-based auto covariance

Nucleosome positioning participates in many cellular activities and plays significant roles in regulating cellular processes (Guo, et al., 2014). Computational methods that can predict nucleosome positioning based on the DNA sequences is highly desired. Here, a computational predictor was constructed by using dinucleotide-based auto covariance and SVMs, and its performance was evaluated by 5-fold cross-validation.

The benchmark data set for the H. sapiens was taken from (Schones, et al., 2008). Since the *H. sapiens* genome and its nucleosome map contain a huge amount of data, according to Liu's strategy (Liu, et al., 2011) the nucleosome-forming sequence samples (positive data) and the linkers or nucleosome-inhibiting sequence samples (negative data) were extracted from chromosome 20 (Guo, et al., 2014). A file named "H_sapiens_pos.fasta" containing 2,273 nucleosome-forming DNA segments is used as the positive dataset, and a file named "H_sapiens_neg.fasta" containing 2,300 nucleosome-inhibiting DNA segments is used as the negative dataset. These two datasets were stored in the folder "repDNA/repDNA/example/", and the source code of this experiment can be found in the Python script file "example2.py" located in "repDNA/repDNA/example/". Of course, the user can also easily find this files in repDNA home-page download page. We employed four third-part Python packages so as to implement the SVM classifiers and plot the ROC curves, including numpy, sklearn, scipy, matplotlib.

First, we should import the DAC class from repDNA.ac module to construct a DAC object, and call make_dac_vec method with parameter *lag*=6 to generate the corresponding feature vectors for the DNA samples in the benchmark dataset:

```
from repDNA.ac import DAC
```

```
# Generate the corresponding feature vectors of samples in the
# dataset.
ac = DAC(lag=6)
pos_vec=ac.make_dac_vec(open('H_sapiens_pos.fasta'),all_property=True)
neg_vec=ac.make_dac_vec(open('H_sapiens_neg.fasta'),all_property=True)
```

The feature vectors of positive and negative samples were stored in two lists pos_vec and neg_vec, respectively. If the format of the input files is not correct, the corresponding error messages will be shown on the screen, and the program will be terminated.

```
print(len(pos_vec))
2273
print(len(neg_vec))
2300
```

Next, we merge the positive and negative feature vectors into a numpy.array (a data type in numpy package), and generate the corresponding labels to represent their classes (1 represents the positive samples, and 0 represents the negative samples).

```
import numpy as np

# Merge positive and negative feature vectors and generate their
# corresponding labels.
vec = np.array(pos_vec + neg_vec)
vec_label = np.array([1] * len(pos_vec) + [0] * len(neg_vec))
```

Next, the SVMs in sklearn package were employed as the classifiers with RBF kernel ($C = 2^{15}$, $gamma = 2^{-9}$). The performance of this predictor was evaluated by using 5-fold cross-validation, and its accuracy will be calculated and shown on the screen.

```
from sklearn import svm
from sklearn import cross_validation

# Use 5-fold cross-validation to evaluate the performance of the
# predictor.
clf = svm.SVC(C= 32768.0, gamma=0.001953125)
scores = cross_validation.cross_val_score(clf, vec, y=vec_label, cv=5)
print('Per accuracy in 5-fold CV:')
print(scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std()*2))
```

```
Per accuracy in 5-fold CV:
[ 0.8568306   0.84590164  0.84153005  0.85886214  0.86870897]
Accuracy: 0.85 (+/- 0.02)
```

Besides, we can also plot the mean ROC curve by using sklearn, scipy and matplotlib packages:

```
from sklearn.cross_validation import StratifiedKFold
from sklearn.metrics import roc_curve, auc
from scipy import interp
import matplotlib.pyplot as plt

# evaluate performance of the predictor by 5-fold cross-validation
# and plot the mean ROC curve.
cv = StratifiedKFold(vec_label, n_folds=5)
classifier = svm.SVC(C=32768.0, kernel='rbf', gamma=0.001953125,
                     probability=True)
mean_tpr = 0.0
mean_fpr = np.linspace(0, 1, 100)
all_tpr = []

for i, (train, test) in enumerate(cv):
    probas_=classifier.fit(vec[train],
                           vec_label[train]).predict_proba(vec[test])
    # Compute ROC curve and area the curve
    fpr, tpr, thresholds = roc_curve(vec_label[test],probas_[:, 1])
    mean_tpr += interp(mean_fpr, fpr, tpr)
    mean_tpr[0] = 0.0

# Plot ROC curve.
plt.plot([0, 1], [0, 1], '--', color=(0.6, 0.6, 0.6), label='Luck')

mean_tpr /= len(cv)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
plt.plot(mean_fpr, mean_tpr, '-',
         label='Mean ROC (area = %0.2f)' % mean_auc, lw=2)

plt.xlim([0, 1.0])
plt.ylim([0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.show()
```

The mean ROC curve of the predictor was shown in **Fig. 2**.

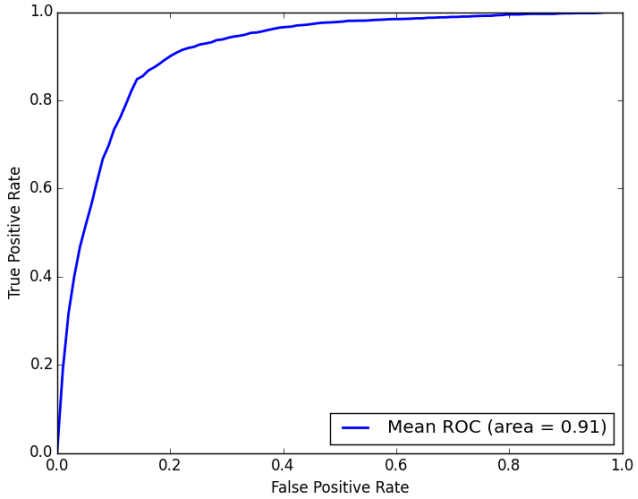


Figure 2. ROC curve achieved by the predictor based on dinucleotide-based auto covariance for predicting the nucleosome positioning in genomes.

2.3 Identifying recombination spots with pseudo dinucleotide composition

Meiotic recombination is an important biological process. Meiotic recombination takes place in some genomic regions (the so-called 'hotspots') with higher frequencies, and in the other regions (the so-called 'coldspots') with lower frequencies (Chen, et al., 2013). So far, the recombination regions have been mainly determined by experiments, which are both expensive and time-consuming. With the avalanche of genome sequences generated in the postgenomic age, it is highly desired to develop automated methods for rapidly and effectively identifying the recombination regions. Here we used the repDNA Python package and some third-party Python package to reproduce the experiments reported in (Chen, et al., 2013). A predictor for recombination spot identification was constructed by combing the pseudo dinucleotide composition and SVM classifiers.

The benchmark data set for the recombination hotspots and coldspots was taken from (Liu, et al., 2012). A file named "hotspots.fasta" containing 490 recombination hotspots was treated as the positive dataset, and a file named "coldspots.fasta" containing 591 recombination coldspots was treated as the negative dataset. These two files were stored in the folder "repDNA/repDNA/example/", and the source code of the experiments reported here can be found in a Python script file "example3.py" located in "repDNA/repDNA/example/". Of course, the user can also easily find this files in repDNA home-page download page. We employed four third-part Python packages so as to implement the SVM classifiers and plot the ROC curves, including numpy, sklearn, scipy, matplotlib.

First, we should import the PseDNC class from repDNA.psenac module to construct a PseDNC object, and call make_psednc_vector method with parameters *lamada*=3, *w*=0.05 (the optimized values of the two parameters as reported in (Chen, et al., 2013)) to generate corresponding feature vectors for the DNA samples in the benchmark dataset:

```
from repDNA.psenac import PseDNC

# Generate the PseDNC feature vectors.
psednc = PseDNC(lamada=3, w=0.05)
pos_vec = psednc.make_psednc_vec(open('hotspots.fasta'))
neg_vec = psednc.make_psednc_vec(open('coldspots.fasta'))
```

The feature vectors of positive and negative samples were stored in two lists pos_vec and neg_vec, respectively. If the format of the input files is not correct, the corresponding error messages will be shown on the screen, and the program will be terminated.

```
Print(len(pos_vec))
490
Print(len(neg_vec))
591
```

Next, we merge the positive and negative feature vectors into a numpy.array (a data type in numpy package), and generate the corresponding labels to represent their classes (1 represents the positive samples, and 0 represents the negative samples):

```
import numpy as np

# Merge positive and negative feature vectors and generate their
# corresponding labels.
vec = np.array(pos_vec + neg_vec)
vec_label = np.array([1] * len(pos_vec) + [0] * len(neg_vec))
```

Next, the SVMs in sklearn package were employed as the classifiers with RBF kernel (C=2^5, *gamma*=0.05 as reported in (Chen, et al., 2013)). The performance of this predictor was evaluated by using 5-fold cross-validation, and its accuracy will be calculated and shown on the screen.

```
from sklearn import svm
from sklearn import cross_validation

# evaluate performance of the predictor by 5-fold cross-validation
# and plot the mean ROC curve.
```

```
clf = svm.SVC(C=32, gamma=0.5)
scores = cross_validation.cross_val_score(clf, vec, y=vec_label, cv=5)
print('Per accuracy in 5-fold CV:')
print(scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std()*2))
```

```
Per accuracy in 5-fold CV:
[ 0.77419355  0.78703704  0.81018519  0.81481481  0.80092593]
Accuracy: 0.80 (+/- 0.03)
```

Besides, we can plot the mean ROC curve by using sklearn, scipy and matplotlib packages:

```
from sklearn.cross_validation import StratifiedKFold
from sklearn.metrics import roc_curve, auc
from scipy import interp
import matplotlib.pyplot as plt

# evaluate performance of the predictor by 5-fold cross-validation
# and plot the mean ROC curve.
cv = StratifiedKFold(vec_label, n_folds=5)
classifier = svm.SVC(C=32, kernel='rbf', gamma=0.5, probability=True)

mean_tpr = 0.0
mean_fpr = np.linspace(0, 1, 100)
all_tpr = []

for i, (train, test) in enumerate(cv):
    probas_=classifier.fit(vec[train],
                           vec_label[train]).predict_proba(vec[test])
    # Compute ROC curve and area the curve
    fpr, tpr, thresholds = roc_curve(vec_label[test], probas_[:, 1])
    mean_tpr += interp(mean_fpr, fpr, tpr)
    mean_tpr[0] = 0.0

# Plot ROC curve.
plt.plot([0, 1], [0, 1], '--', color=(0.6, 0.6, 0.6), label='Luck')

mean_tpr /= len(cv)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
plt.plot(mean_fpr, mean_tpr, '-',
         label='ROC (area = %0.2f)' % mean_auc, lw=2)

plt.xlim([0, 1.0])
```

```
plt.ylim([0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.show()
```

The plotted mean ROC curve of the predictor was shown in **Fig. 3**.

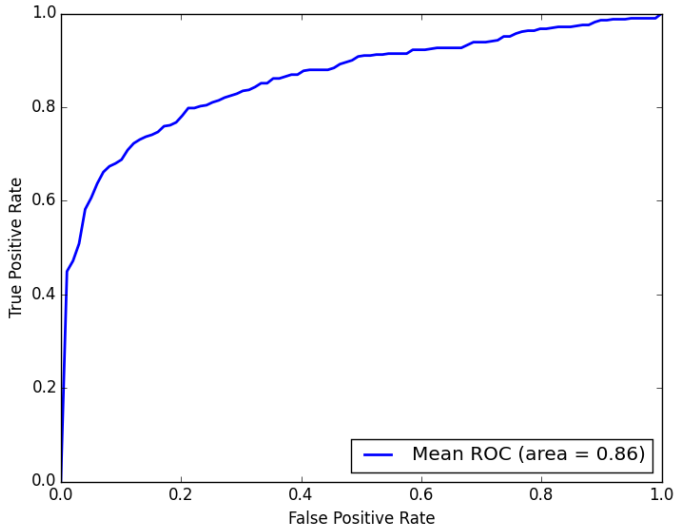


Figure 3. ROC curve achieved by the predictor based on pseudo dinucleotide composition for predicting recombination spots.

Reference

Chen, W., *et al.* iRSpot-PseDNC: identify recombination spots with pseudo dinucleotide composition. *Nucleic acids research* 2013;41(6):e68.

Gross, D.S. and Garrard, W.T. Nuclease hypersensitive sites in chromatin. *Ann. Rev. Biochem* 1998;57:159-197.

Guo, S.-H., *et al.* iNuc-PseKNC: a sequence-based predictor for predicting nucleosome positioning in genomes with pseudo k-tuple nucleotide composition. *Bioinformatics* 2014;30(11):1522-1529.

Hunter, J.D. Matplotlib: A 2D Graphics Environment. *Computing in Science and Engineering* 2007;9(3):90-95.

Jones E, O.E., Peterson P, et al. SciPy: Open Source Scientific Tools for Python. 2001.

Li, Q.L., *et al.* Locus control regions. *Blood* 2002;100(9):3077–3086.

Liu, G., *et al.* Sequence-dependent prediction of recombination hotspots in Saccharomyces cerevisiae. *Journal of theoretical biology* 2012;293:49-54.

Liu, H.D., *et al.* Analysis of nucleosome positioning determined by DNA helix curvature in the human genome. *BMC Genomics* 2011;12(72).

Noble, W.S., *et al.* Predicting the in vivo signature of human gene regulatory sequences. *Bioinformatics* 2005;21 Suppl 1:i338-343.

Pedregosa, F., *et al.* Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*

2011;12:2825-2830.

Schones, D.E., *et al.* Dynamic Regulation of Nucleosome Positioning in the Human Genome. *Cell* 2008;10:887-898.

Walt, S.v.d., Colbert, S.C. and Varoquaux, G. The NumPy array: a structure for efficient numerical computation. *Computing in Science and Engineering* 2011;13:22-30.