COSI123A HW5

HyunJae Pi

November 12, 2020

Two datasets are provided.
(a) The training set (training.data): Each line in training data is a sample containing a class label and the features that are separated by a comma. For example,

1,GTCTCTCTCTCTCTCTCTCTTTCTCTGCAGGTTCTCCCCATGACACCACCTGAACGTCTC

(b) The test set (test.data): Each line is a sample without its class label.

Task: Use the training set to build a classifier (limited to what we have covered in our class), and apply it to the test set and submit your prediction results in a text file (one prediction per line). The order of your predictions should be the same to the order of test samples. In addition, you should also submit a written report explaining what you do to get your results, which should have enough details for readers to reproduce your results. Grading: Written report [50 points]. We will assume that the prediction accuracy follows a Gaussian-like distribution. The mean and variance of all predictions will be first calculated and be used to calculate your accuracy scores. Those higher than (mean + 2 std) will get 50 points, those equal or lower than (mean - 3std) or random guess will get 25 points, and the rest in between will get a score based on the scale decided by the first two settings. Tips: A referenced value of 10-fold cross-validation accuracy on train dataset is 0.91. It is also possible to improve the accuracy to 0.95.

Late penalty 15 points off.
Deadline: 11/10 23:59pm.
Cutoff date: 11/12 23:59pm.

1. Preprocess data

Assuming this as DNA sequence data, there are many proprocessing techniques available. Among these, I tried three approaches that convert sequence data to numeric values.

1) Integer representation

It simply converts 'A' to 1, 'C' to 2, 'G' to 3, 'T' to 4.

2) Electron-ion interaction pseudopotential (EIIP) representation

It is proposed by Nair and Sreenadhan. Briefly, EIIP values of nucleotides to represent biological sequences and to locate exons. A numerical sequence represents the distribution of free electron energies can be called "EIIP indicator sequence". This representation converts 'G' to 0.0806, 'A' to 0.1260, 'C' to 0.1340, 'T' to 0.1335.

(reference) A. S. Nair, S. P. Sreenadhan, A coding measure scheme employing electron-ion interaction pseudopotential (eiip), Bioinformation 1 (6) (2006) 197.

3) Kmer representation

Kmer refers to a sub-sequence of length k contained within a sequence. In bioinformatics, for instance, when we have a sequence of 'ACCTAGCTG', we have 3 3-mers (ACC, TAG, CTG). This representation allows to implement information contained in a group of subsequences. Although the idea is fascinating and it is popularly used , it didn't work well in the training data.

I tested these three representations with different classifiers that I used in this report. For this dataset, EIIP was chosen because it gave the best accuracy regardless of classifiers.

2. Test different classifiers

SVM, Decision Trees, AdaBoost and Bagging were tested. All classifiers gave reasonably good accuracy (around 0.9) without fine tuning of parameters. However, Decision Trees was better than SVM both in single simulations or with ensemble approaches. After parameter tuning, the accuracies of AdaBoost with Decision Trees and Bagging with Decision Trees were 0.931 and 0.942, respectively. Therefore, Bagging with Decision Trees was chosen because it gave the best accuracy 0.942 (cross validation = 5, 31 base learners). The following table summarizes the simulations that I run with different classifiers and best parameters.

Classifier / kernel	***Accuracy***	***Parameter 1***	***Parameter 2***
SVM / rbf	0.905	C = 1	gamma = 0.01
SVM / polynomial	0.886	C = 0.001	gamma = 1
SVM / linear	0.897	C = 10	N/A
Decision Trees	0.924	max_depth = 8	min_samples_split = 0.01
AdaBoost w/ Decision Trees	0.927	n_estimators = 35	learning rate = 0.6
AdaBoost w/ SVM	0.931	n_estimators = 40	learning rate = 0.9
Bagging w/ Decision Trees	**0.942**	**n_estimators = 31**	N/A
Bagging w/ SVM	0.898	n_estimators = 33	N/A

3. Predict classes from test data

A classifier was built with Bagging (base estimator: Decision Trees with 31 learners) using training data and the prediction was generated from test data. The result can be found in **'hw5_prediction.txt.'**

4. Codes

Here are the codes that I used for hw5.

1) 'hw5_preprocess.py' converts sequence data into numberic representation.
2) 'hw5_find_best_classifier.py' test different classifiers and parameters and identifies the best one.
3) 'hw5_run_test.py' builds the best classifier from training data and predicts classes of test data.

hw5_preprocess.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 12 11:26:06 2020

hw5_preprocess.py

This program converts letters to numeric values according to EIIP representation.
    G   to 0.0806,   A   to 0.1260,   C   to 0.1340,   T   to 0.1335

@author: HyunJae Pi, hyunpi@brandeis.edu
"""

import numpy as np
import pandas as pd
import re
from sklearn.preprocessing import LabelEncoder

# convert string to array
def string_to_array(my_string):
    my_string = my_string.lower()
    my_string = re.sub('[^acgt]', 'z', my_string)
    my_array = np.array(list(my_string))
    return my_array

# convert letter to number (EIIP representation -- see hw5 report)
def letter_to_number(my_array):
    tmp = label_encoder.transform(my_array)
    num = tmp.astype(float)
    num[num == 0] = .1260 # A
    num[num == 1] = .1340 # C
    num[num == 2] = .0806 # G
    num[num == 3] = .1335 # T
    num[num == 4] = .00 # anything else
    return num

# label encoder
label_encoder = LabelEncoder()
label_encoder.fit(np.array(['a','c','g','t','z']))

# 1. convert training data
df0 = pd.read_csv("./training.data", header=None)
df0.columns=["class", "seq_letter"]
n_rows0 = df0.shape[0]
n_cols0 = df0.shape[1]


tmp_data0 = np.zeros((n_rows0, 60))
for i in range(0, n_rows0):
    tmp_data0[i, :] = letter_to_number(string_to_array(df0.loc[i,'seq_letter'])).T
print(tmp_data0.shape)

tmp_class = np.array(df0["class"])
training_data = np.column_stack((tmp_data0, tmp_class))

# save
df_save = pd.DataFrame(training_data)
df_save.to_csv('training2b.csv', index=False, header=None)

# 2. convert test data
df1 = pd.read_csv("./test.data", header=None)
df1.columns=["seq_letter"]

n_rows1 = df1.shape[0]
n_cols1 = df1.shape[1]
tmp_data1 = np.zeros((n_rows1, 60))
for i in range(0, n_rows1):
    tmp_data1[i, :] = letter_to_number(string_to_array(df1.loc[i,'seq_letter']))
print(tmp_data1.shape)

# save
df_save = pd.DataFrame(tmp_data1)
df_save.to_csv('test2b.csv', index=False, header=None)
```

hw5_find_best_classifier.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 11 07:07:29 2020

hw5_find_best_classifier.py

Using preprocessed training data, this program
tests different classifiers (SVM, Decision Trees, AdaBoost, and Bagging)
and identifies the best one.

@author: HyunJae Pi, hyunpi@brandeis.edu
"""

import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

from sklearn import preprocessing
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier, BaggingClassifier

# select_params in 1D or 2D parameter space
def select_params(X, y, clf, param1_name, param1, param2_name ="", param2 =[], nfolds = 5):
    if param2_name == "":
        param_grid = {param1_name: param1}
    else:
        param_grid = {param1_name: param1, param2_name : param2}
    grid_search = GridSearchCV(clf, param_grid, cv=nfolds)
    grid_search.fit(X, y)
    grid_search.best_params_
    return grid_search.best_params_


# 0. load a preprocessed training data
df0 = pd.read_csv("./training2b.csv", header=None)
n_features = df0.shape[1]-1
X = preprocessing.scale(df0.loc[:, 0:n_features-1].values)
```

```
y = df0.loc[:, n_features].values

# 1. SVM w/ rbf
clf = svm.SVC(kernel = 'rbf')
param1 = [0.001, 0.01, 0.1, 1, 10] # C
param2= [0.001, 0.01, 0.1, 1] # gamma
best_params = select_params(X, y, clf, 'C', param1, 'gamma', param2)
print("1. SVM with rbf")
clf_svm = svm.SVC(kernel = 'rbf', C = best_params['C'], gamma = best_params['gamma']) #0.905
print("accuracy: %.3f" %(np.mean(cross_val_score(clf_svm, X, y, cv = 5)))) # {'C': 1, 'gamma
print(best_params)

# 2. SVM w/ polynomial
clf = svm.SVC(kernel = 'poly')
param1 = [0.0001, 0.001, 0.01, 0.1, 1] # C
param2= [0.01, 0.1, 1, 10] # gamma
best_params = select_params(X, y, clf, 'C', param1, 'gamma', param2)
print("2. SVM with polynomial")
clf_svm = svm.SVC(kernel = 'poly', C = best_params['C'], gamma = best_params['gamma'])
print("accuracy: %.3f" %(np.mean(cross_val_score(clf_svm, X, y, cv = 5)))) # 0.886
print(best_params) # {'C': 0.0001, 'gamma': 1}

# 3. SVM w/ linear
clf = svm.SVC(kernel = 'linear')
param1 = [ 0.1, 1, 10, 100] # C
param2= [] # gamma
best_params = select_params(X, y, clf, 'C', param1)
print("3. SVM with linear")
clf_svm = svm.SVC(kernel = 'linear', C = best_params['C'])
print("accuracy: %.3f" %(np.mean(cross_val_score(clf_svm, X, y, cv = 5)))) #0.897
print(best_params) # {'C': 10}


# 4. Decision Tree
clf = DecisionTreeClassifier(random_state=0,  criterion='gini',splitter='best')
max_depths = np.linspace(1, 10, 10, endpoint=True)
min_samples_splits = np.linspace(0.01, 0.2, 10, endpoint=True)
best_params = select_params(X, y, clf, 'max_depth', max_depths, 'min_samples_split', min_sam
print("4. Decision Trees\n")
clf_dt = DecisionTreeClassifier(random_state=0,  criterion='gini',splitter='best', max_depth
print("accuracy: %.3f" %(np.mean(cross_val_score(clf_dt, X, y, cv = 5)))) #0.924
print(best_params) #{'max_depth': 8.0, 'min_samples_split': 0.01}
```

```
##### ensemble classifiers
# base estimators
svc = svm.SVC(probability=True, kernel='linear')
dt = DecisionTreeClassifier()
n_estimators = np.linspace(30, 40, 10, endpoint = True, dtype = int)
learning_rate = np.linspace(0.1, 1, 10, endpoint = True)

# 5. AdaBoost w/ decision trees
clf =  AdaBoostClassifier(base_estimator=dt)
best_params = select_params(X, y, clf, 'n_estimators', n_estimators, 'learning_rate', learnin
print("5. AdaBoost w/ Decision Trees")
clf_ab_t = AdaBoostClassifier(base_estimator=dt, n_estimators = best_params['n_estimators'],
print("accuracy: %.3f" %(np.mean(cross_val_score(clf_ab_t, X, y, cv = 5)))) # 0.927
print(best_params) # 0.6 & 35

# 6. AdaBoost w/ svm -- too slow later
clf =  AdaBoostClassifier(base_estimator=svc)
best_params = select_params(X, y, clf, 'n_estimators', n_estimators, 'learning_rate', learnin
print("6. AdaBoost w/ SVM")
clf_ab_s = AdaBoostClassifier(base_estimator=svc, n_estimators = best_params['n_estimators'],
print("accuracy: %.3f" %(np.mean(cross_val_score(clf_ab_s, X, y, cv = 5)))) #0.931
print(best_params) #0.9 40

svc = svm.SVC(kernel='linear')

# 7. Bagging w/ decision trees
clf = BaggingClassifier(base_estimator=dt, random_state=1)
best_params = select_params(X, y, clf, 'n_estimators', n_estimators)
print("7. Bagging w/ Decision Trees")
clf_bag_t = BaggingClassifier(base_estimator=dt, n_estimators = best_params['n_estimators'])
print("accuracy: %.3f" %(np.mean(cross_val_score(clf_bag_t, X, y, cv = 5)))) #0.942
print(best_params) # 31

# 8. Bagging w/ svm
clf = BaggingClassifier(base_estimator=svc, random_state=1)
best_params = select_params(X, y, clf, 'n_estimators', n_estimators)
print("8. Bagging w/ SVM")
clf_bag_t = BaggingClassifier(base_estimator=svc, n_estimators = best_params['n_estimators'])
print("accuracy: %.3f" %(np.mean(cross_val_score(clf_bag_t, X, y, cv = 5)))) #0.898
print(best_params) # 33
```

hw5_run_test.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 12 11:03:43 2020

hw5_run_test.py

This program runs the identified best classifier on the test dataset
Bagging w/ Decision Trees (31 estimators)

@author: HyunJae Pi, hyunpi@brandeis.edu
"""

import numpy as np
import pandas as pd
from sklearn import preprocessing#from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier


# training data
df0 = pd.read_csv("./training2b.csv", header=None)
n_features = df0.shape[1]-1
X_training = preprocessing.scale(df0.loc[:, 0:n_features -1].values)
y_training = df0.loc[:, n_features].values

# test data
df1 = pd.read_csv("./test2b.csv", header=None)
X_test = preprocessing.scale(df1.loc[:, 0:n_features -1].values)

clf = BaggingClassifier(base_estimator = DecisionTreeClassifier(), n_estimators = 31).fit(X_t
y_test = clf.predict(X_test).astype(int)

# save
np.savetxt('./hw5_prediction.txt', y_test, fmt = '%d')
```