یادگیری ماشین

تمرین شماره 2

Ensemble Algorithms

حسین توکلیان

شماره دانشجویی

9860571

خرداد 1399

دکتر ستار هاشمی

کتابخانه های مورد استفاده:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_fscore_support, roc_auc_score
from imblearn.metrics import geometric_mean_score
import numpy as np
from imblearn.over_sampling import SMOTE, RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
```

خواندن فایل

```
def load_data(file_name):
    df = pd.read_csv(file_name)
    df = df.fillna(-1)
    x = df.iloc[:, :-1].values
    y = df.iloc[:, -1].values
```

جایگذاری مقادیر خالی با مقدار -1

```
df = df.fillna(-1)
```

جداسازی:

مقادیر ویژگیها

```
x = df.iloc[:, :-1].values
```

مقادیر کلاس ها

```
y = df.iloc[:, -1].values
```

پیاده سازی مربوط به کلاسیفایرهای بخش Adaboost و Bagging

```
def report_measures_for_clf(x, y, clf, sampler):
```

این تابع x و y را دریافت میکند بعد یک کلاسیفایر دریافت میکند و یک Sampler که سمپلر میتواند smote , under sampler و یا oversampler باشد.

```
trn_x, tst_x, trn_y, tst_y = train_test_split(x, y, train_size = trn_ratio, stratify = y)
```

با استفاده از تابع train_test_split مربوط به کتابخانه sklearn داده ها را به test و train تقسیم میکنم.

با استفاده از sampler گفته شده داده ها را نمونه گیری کردم.

```
clf.fit(s_x, s_y)
```

کلاسیفایر را روی داده ها fit کردم

```
p = clf.predict_proba(tst_x)
```

در نهایت روی داده های test احتمال هر کدام از کلاس ها را بدست آوردم.

```
prd = np.argmax(p, axis=1)
prd[prd == 0] = -1
```

و بعد لیبل کلاسهای prd را بدست آوردم و جاهایی که لیبل صفر شده مقدار 1- قرار میدهم.

```
prc, recall, fscore, _ = precision_recall_fscore_support(tst_y, prd, labels=[1])
```

با استفاده از این تابع مقدادیر precision , recall , fscore را بدست می آورم

```
auc = roc_auc_score(tst_y, p[:, 1])
```

مقدار auc را بدست می آورم

```
g_mean = geometric_mean_score(tst_y, prd)
```

و در نهایت مقدار g_mean

```
clfs = {'NB': GaussianNB(), 'Linear-SVM': SVC(kernel='linear', probability=True),
    'RBF-SVM': SVC(probability=True), '1NN': KNeighborsClassifier(n_neighbors=1)}
```

سا خت یک دیکشنری برای ذخیره کلاسیفایرها

```
samplers = {'SMOTE': SMOTE(), 'UnderSampler': RandomUnderSampler(), 'OverSampler':
RandomOverSampler()}
```

sampler ها با استفاده از کتابخانه imblearn

```
file_name = 'Covid-19.csv'
trn_ratio = 0.7
```

خواندن داده ها به نسبت 70 درصد train

```
n_itr = 10
```

تعداد تکرار برای هر کدام از موارد و میانگین بگیریم

```
for clf_name in clfs:
  for sampler_name  in samplers:
```

برای هر کدام از کلاسیفیرها و نمونه گیرها کاری که انجام شده این هست که

```
for i in range(n_itr):
  p, r, f, a, g = report_measures_for_clf(x, y, clfs[clf_name], samplers[sampler_name])
```

به اندازه 10 بار (n_itr) تابع report_measures را صدا کردم و ورودی داده ها کل داده ها بوده که تقسیم به train و test شده کلاسیفایر و sampler نیز به آن داده شده و مقادیر p=precision, r=recall, f= fscore, a=auc, g=gmean را حساب کرده و برگردانده.

```
prc += p
recall += r
fscore += f
auc += a
g_mean += g
```

و آن مقادیر را با مقادیر قبلی جمع کرده

```
prc /= n_itr
recall /= n_itr
fscore /= n_itr
auc /= n_itr
g_mean /= n_itr
```

و در نهایت به تعداد تکرار تقسیم کردم تا مقدار میانگین بدست بیاد

```
print('{}, {}, Precision: {:.2f}, Recall: {:.2f}, F-Score: {:.2f}, AUC: {:.2f}, G-Mean: {:.2f}'.format(clf_name, sampler_name, prc, recall, fscore, auc, g_mean))
```

و در آخر نیز مقدار خروجی چاپ شده

NB, SMOTE, Precision: 0.19, Recall: 0.86, F-Score: 0.31, AUC: 0.81, G-Mean: 0.81

NB, UnderSampler, Precision: 0.18, Recall: 0.90, F-Score: 0.30, AUC: 0.81, G-Mean: 0.80

NB, OverSampler, Precision: 0.08, Recall: 0.94, F-Score: 0.15, AUC: 0.82, G-Mean: 0.47

==========

Linear-SVM, SMOTE, Precision: 0.21, Recall: 0.80, F-Score: 0.33, AUC: 0.86, G-Mean: 0.80

Linear-SVM, UnderSampler, Precision: 0.20, Recall: 0.89, F-Score: 0.33, AUC: 0.88, G-Mean: 0.82

Linear-SVM, OverSampler, Precision: 0.20, Recall: 0.85, F-Score: 0.32, AUC: 0.86, G-Mean: 0.81

==========

RBF-SVM, SMOTE, Precision: 0.28, Recall: 0.69, F-Score: 0.40, AUC: 0.89, G-Mean: 0.78

RBF-SVM, UnderSampler, Precision: 0.19, Recall: 0.94, F-Score: 0.31, AUC: 0.89, G-Mean: 0.82

RBF-SVM, OverSampler, Precision: 0.27, Recall: 0.75, F-Score: 0.40, AUC: 0.90, G-Mean: 0.80

==========

NN, SMOTE, Precision: 0.28, Recall: 0.55, F-Score: 0.37, AUC: 0.73, G-Mean: 0.701

NN, UnderSampler, Precision: 0.20, Recall: 0.82, F-Score: 0.32, AUC: 0.80, G-Mean: 0.801

NN, OverSampler, Precision: 0.29, Recall: 0.48, F-Score: 0.36, AUC: 0.70, G-Mean: 0.661

==========

Bagging

```
def get_a_bootstrap(x, y):
```

ساخت bootstrap از داده های کلاس ماینر و کلاس میجر bootstrap میگیریم و کنار هم قرار میدهیم.

```
mjr_x = x[y == -1]
mjr_y = y[y == -1]

mnr_x = x[y == 1]
mnr_y = y[y == 1]
```

جدا سازی کلاس ماینر از میجر

```
n_mnr = mnr_x.shape[0]
n_mjr = mjr_x.shape[0]
```

بدست آوردن تعداد داده های کلاس ماینر و میجر

```
r_mnr = np.random.randint(0, n_mnr, n_mnr)
r_mjr = np.random.randint(0, n_mjr, n_mnr)
```

به تعداد داده های ماینر و میجر عدد تصادفی تولید کردم.

```
x_bs = np.concatenate((mjr_x[r_mjr], mnr_x[r_mnr]), axis = 0)
y_bs = np.concatenate((mjr_y[r_mjr], mnr_y[r_mnr]), axis = 0)
```

به تعداد مساوی sample گرفته و آنها را به هم چسباندم

```
def train_bagging(n_trees, x, y):
    x, y = get_a_bootstrap(x, y)
    tree_list = []
```

یک bootstrap بگیرد و با استفاده از آن یک درخت را train کند

لیست درخت را داریم که در ابتدا خالی است

```
for _ in range(n_trees):
    tree = DecisionTreeClassifier()
    tree.fit(x, y)
    tree_list.append(tree)
```

و به تعداد درختها درخت میسازم

درخت را روی دادهای جدید ارسال شده به عنوان داده آموزشی fit میکنم

اون درخت جدید را به لیست درختهای قبلی اضافه میکنم.

```
def predict_bagging(tree_list, x):
    p = 0
```

لیستی از درخت ها و یکسری داد (x) را دریافت میکند

```
p += tree.predict_proba(x)
```

برای هر کدام از درختها احتمال تعلق به هر کلاس را حساب میکند.

```
p /= len(tree_list)
return p
```

احتمالات را نرمال میکنم عددی بین 0-1 برای همه کلاسها و آن احتمالات را به عنوان خروجی بر میگردانم.

```
def report_measures_for_bagging(x, y, n_trees, trn_ratio):
```

دریافت کل داده ها ، تعداد درختهایی که هر کدام از baggingها باید داشته باشد و درصد داده ها

```
trn_x, tst_x, trn_y, tst_y = train_test_split(x, y, train_size = trn_ratio, stratify = y)
bag_trees = train_bagging(n_trees, trn_x, trn_y)
p = predict_bagging(bag_trees, tst_x)
```

با استفاده از تابع train_test_split مربوط به کتابخانه sklearn داده ها را به test و train تقسیم میکنم.

با استفاده از این داده ها bagging را آموزش میدهد.

و احتمال تعلق به کلاسها رو با استفاده از predict_bagging حساب میکند

```
prd = np.argmax(p, axis=1)
prd[prd == 0] = -1
```

لیبل ها حساب میشود با استفاده از اینکه احتمال کدام کلاس بیشتر بوده

```
prc, recall, fscore, _ = precision_recall_fscore_support(tst_y, prd, labels=[1])
auc = roc_auc_score(tst_y, p[:, 1])
g_mean = geometric_mean_score(tst_y, prd)
return prc[0], recall[0], fscore[0], auc, g_mean
```

مقادیر گفته شده حساب شده و به عنوان خروجی بر میگردد.

```
file_name = 'Covid-19.csv'
trn_ratio = 0.7
```

خواندن داده ها به نسبت 70 درصد train

```
n_itr = 10
```

تعداد تکرار برای هر کدام از موارد و میانگین بگیریم

```
for n_trees in [11, 31, 51, 101]:
    prc, recall, fscore, auc, g_mean = (0, 0, 0, 0, 0)
```

تعداد درختها را این مقادیر قرار دهیم.

```
p, r, f, a, g = report_measures_for_bagging(x, y, n_trees, trn_ratio)
```

تابع report_mesure را صدا میزنیم

Bagging, #Trees: 11, Precision: 0.20, Recall: 0.68, F-Score: 0.31, AUC: 0.79, G-Mean: 0.74

Bagging, #Trees: 31, Precision: 0.22, Recall: 0.73, F-Score: 0.33, AUC: 0.82, G-Mean: 0.77

Bagging, #Trees: 51, Precision: 0.19, Recall: 0.72, F-Score: 0.30, AUC: 0.80, G-Mean: 0.75

Bagging, #Trees: 101, Precision: 0.22, Recall: 0.69, F-Score: 0.33, AUC: 0.80, G-Mean: 0.76

Adaboost

```
def get_a_balanced_set(x, y):
    mjr_x = x[y == -1]
    mjr_y = y[y == -1]

    mnr_x = x[y == 1]
    mnr_y = y[y == 1]
```

داده هی مثبت را در نظر گرفته و مثبت ها را ار منفی ها جدا کرده

```
ind = [i for i in range(mjr_x.shape[0])]
r = np.random.choice(ind, size=mnr_x.shape[0], replace=False)
```

اندیس تعدادی از داده های ماینر را مساوی تعدادی از داده های میجر انتخاب میکند به صورت غیر تکراری

```
x_bs = np.concatenate((mjr_x[r], mnr_x), axis = 0)
y_bs = np.concatenate((mjr_y[r], mnr_y), axis = 0)

return x_bs, y_bs
```

و آنها را به هم میچسباند و داده x , y را برایمان میسازد. ورودی x و خروجی y

```
def train_adaboost(x, y, n_trees, max_depth):
```

تابع train_adaboost را داریم که فقط یک adaboost را برایمان train میکند.

تعداد داده های آموزشی(x,y) تعداد درخت و عمق درخت

```
n_samples = x.shape[0]
```

تعداد داده های آموزشی را بدست آوردم

```
d = np.ones((n_samples,)) / n_samples
```

مقدار اولیه d را ساختم که ضریب هر کدام از داده ها را نشان میدهد که در ابتدا همه با هم برابر 1 است

```
tree_list = []
alpha = np.zeros((n_trees))
```

لیست درختها و alpha ضریب هر درخت

```
inds = [i for i in range(n_samples)]
```

اندیس تمام داده ها از 0 تا n

```
for t in range(n_trees):
    sel_inds = np.random.choice(inds, size=n_samples, replace=True, p=d)
```

در طول تکرار ها کاری که انجام میدم به صورت تصادفی داده هایی را انتخاب میکنم با توزیع d و اجازه جایگذاری هم وجود داره برای اینکه هر داده ای که احتمال بیشتری داشته باشد نمونه های بیشتری از آن انتخاب خواهد شد.

```
tree = DecisionTreeClassifier(max_depth=max_depth)
tree.fit(x[sel_inds], y[sel_inds])
```

با استفاده از داده های انتخاب شده یک DecisionTreeClasifier ایجاد میشود و با استفاده از اون داده ها آموزش داده میشود.

```
l = tree.predict(x)
match = (l != y)
err = (d * match).sum()
```

لیبل داده ها بدست می آید

آنهایی که لیبل یکسان ندارند را بدست میاریم

ضرب در d کرده و جمع خطا را بدست می آوریم.

```
if err > 0.5:
   return False
```

اگر جمع از 5. بیشتر شد از تابع خارج میشیم که دوباره تابعی که آن را صدا زده باید عمق بیشتری به درخت بدهد تا بتواند کار ادامه یابد. چون خطای بیشتر از نصف کلاسیفایر درست پیشبینی نمیکند.

```
alpha[t] = 0.5 * np.log((1 - err) / err)
tree_list.append(tree)
```

مقدار alpha تکرار بدست میاد و درخت به لیت درختها اضافه میشود.

```
d = d * np.exp(-alpha[t] * l * y)
d /= d.sum()

return tree_list, alpha
```

update مقدارd بر اساس فرمول در اسلایدها و نرمال شدن برای داشتن حالت توزیع احتمالی و

درنهایت باز گرداندن لیست درخت احتمالی و alpha

```
def predict_adaboost(tree_list, alpha, x):
```

تابع predict_adaboost لیست درخت ها، آلفا و چیزی که باید براش پیشبینی ها را انجام دهیم دریافت میکند.

```
s = alpha[0] * tree_list[0].predict(x)
for i in range(1, len(tree_list)):
   s += alpha[i] * tree_list[i].predict(x)
```

پیشبینی درختها ضرب در آلفا و جمع با s

```
label = np.sign(s)
p = 1. / (1 + np.exp(-2 * s))
return label, p
```

در نهایت لیبل میشود علامت s برای هر کدام از داده ها

و چون احتمال را نیاز داریم با استفاده از تابع لاجستیک رگرسیون تبدیل به مقدار احتمال میکنمیم خروجی کلاسیفایر s را.

و داده ها را به عنوان خروجی باز میگردانیم.

```
def report_measures_for_adaboost(x, y, n_trees, n_ensembles, trn_ratio):
```

معیارها را در adaboost حساب میکنیم.

x,y داده های آموزشی n_tree تعداد درخت n_ensembles تعداد adoboost هایی که ساخته میشوند که در صورت تمرین n_trees=S و n_ensembles = T است و تعداد داده های آموزشی trn_ratio

```
trn_x, tst_x, trn_y, tst_y = train_test_split(x, y, train_size = trn_ratio, stratify = y)
all_trees = []
all_alpha = []
```
تقسیم داده ها

باید تمام ensemble ها را ایجاد کرد و درختهاش رو کنار هم قرار داد و با استفاده از آن پیشبینی را انجام داد

```
for _ in range(n_ensembles):
  bs_x, bs_y = get_a_balanced_set(trn_x, trn_y)
  max_depth = 1
```
بنابر این به تعداد n_ensembles بار کاری که انجام میدم

get_a_balanced از داده ها آموزشی میگیرم

و مقدار عمق را 1 قرار میدهم و سعی میکنم ببینم با این عمق کار به درستی انجام میشود یا نه

```
while True:
  out = train_adaboost(bs_x, bs_y, n_trees, max_depth)
  if out:
    ada_trees, alpha = out
    break
```
یعنی با عمق 1 adoboost با آن داده هایی که بالانس هستند (bs_x,bs_y,...) آموزش میدم اگر درست درآمد یعنی flase برگشت نشد هم درخت ها و هم آلفا را خواهم داشت

و از حلقه خارج میشوم

```
max_depth += 1
```
وگرنه به مقدار عمق یکی اضافه میکنم و این کار را تکرار میکنم تا عمق کافی باشد و این خطای درختم از 50 درصد بیشتر نشود.

```
if max_depth > 10:
```
اگر عمق را تا مقدار 10 افزایش دادم و اتفاقی نیفتاد

```
bs_x, bs_y = get_a_balanced_set(trn_x, trn_y)
  max_depth = 1
```
سعی میکنم داده آموزشی ام را عوض کنم شاید اشکال ازآن باشد عمق را 1 قرار میدهم و حلقه را دوباره تکرار میکنم.

```
all_trees.extend(ada_trees)
all_alpha.extend(alpha)
```
در نهایت درخت های ایجاد شده را به لیست درختها اضافه میکنم و alpha را هم اضافه میکنم.

```
prd, p = predict_adaboost(all_trees, all_alpha, tst_x)
```
تابع predict_adoboost را صدا میزنم با تمامی درختهای ensembles ها و آلفاها و داده های تست(tst_x) را بش میدم و مقادیر خروجی را برام حساب میکنه

```
prc, recall, fscore, _ = precision_recall_fscore_support(tst_y, prd, labels=[1])
auc = roc_auc_score(tst_y, p)
g_mean = geometric_mean_score(tst_y, prd)
return prc[0], recall[0], fscore[0], auc, g_mean
```
و این مقادیر(precision,recall,fscore,auc,g_mean) را نیز حساب کرده و باز میگرداند

```
file_name = 'Covid-19.csv'
trn_ratio = 0.7
x, y = load_data(file_name)
n_itr = 10
```

```
for ensemble_size in [10, 15]:
    for n_trees in [11, 31, 51, 101]:
```

اشاره شده بود مقادیر ensemble ها را 10 و 15 قرار دهیم و مقادیر هر کدام 11,31,51,101

```
prc, recall, fscore, auc, g_mean = (0, 0, 0, 0, 0)
for i in range(n_itr):
    p, r, f, a, g = report_measures_for_adaboost(x, y, n_trees, ensemble_size, trn_ratio)
    prc += p
    recall += r
    fscore += f
    auc += a
    g_mean += g

prc /= n_itr
recall /= n_itr
fscore /= n_itr
auc /= n_itr
g_mean /= n_itr
```

```
print('ensemble-size: {}, #Trees: {}, Precision: {:.2f}, Recall: {:.2f}, F-Score: {:.2f}, AUC: {:.2f}, G-Mean:
{:.2f}'.format(ensemble_size, n_trees, prc, recall, fscore, auc, g_mean))
```

اندازه ensemble و دیگر مقادیر چاپ میشوند.

ensemble-size: 10, #Trees: 11, Precision: 0.19, Recall: 0.96, F-Score: 0.32, AUC: 0.90, G-Mean: 0.83

ensemble-size: 10, #Trees: 31, Precision: 0.19, Recall: 0.93, F-Score: 0.32, AUC: 0.90, G-Mean: 0.83

ensemble-size: 10, #Trees: 51, Precision: 0.20, Recall: 0.91, F-Score: 0.33, AUC: 0.89, G-Mean: 0.83

ensemble-size: 10, #Trees: 101, Precision: 0.21, Recall: 0.90, F-Score: 0.34, AUC: 0.88, G-Mean: 0.83

ensemble-size: 15, #Trees: 11, Precision: 0.19, Recall: 0.95, F-Score: 0.31, AUC: 0.89, G-Mean: 0.83

ensemble-size: 15, #Trees: 31, Precision: 0.20, Recall: 0.94, F-Score: 0.34, AUC: 0.89, G-Mean: 0.84

ensemble-size: 15, #Trees: 51, Precision: 0.19, Recall: 0.91, F-Score: 0.32, AUC: 0.87, G-Mean: 0.82

Questions:

- Why should we set max_depth parameter in AdaBoost.M1 so that the base classifiers become a little better than random guess?

Because if the depth of tree > 1 then we do not consider it as the weak classifier.

- What do we mean by stable, unstable, and weak classifier?

Stable learning algorithm is one for which the prediction does not change much when the training data is modified slightly(not sensitive to the noise). But unstable don't act like this. Stable classifiers do not change much over replicates of T.

Some classification and regression methods are unstable in the sense that small perturbations in their training sets or in construction may result in large changes in the constructed predictor. Subset selection methods in regression, decision trees in regression and classification, and neural nets are unstable.

When we used AdaBoost, my weak classifiers were basically thresholds for each data attribute. Those thresholds need to have a performance of more than 50%, if not it would be totally random. A weak classifier is simply a classifier that performs poorly, but performs better than a random guessing.

- What kind of classifiers should be used in Bagging? How about AdaBoost.M1? Why?

We can bagging with any base learner but they should tend to have a low bias and, consequently, high variance.

AdaBoost is best used to boost the performance of decision trees on binary classification problems.

They are simpler in computation.