Многоклассовая классификация

Задача классификации

Выборка: $X^l = \{x_1, \dots, x_l\}$

Ответы: $y_i \in \{0,1\}, i = 1, \dots, l$

Алгоритм: $a: X \to Y$

Один против всех (one-vs-all)

Идея: построить K классификаторов, отделяющих каждый класс от остальных.

Получим K задач бинарной классификации:

- Объекты: $X^k = X^l$;
- Ответы: $y_i^k = [y_i = k]$;
- Оценка принадлежности: $b_k(x) \in R$.

Итоговый алгоритм:

$$a(x) = \underset{k=1,\dots,K}{\operatorname{argmax}}\, b_k(x)$$

Все против всех (all-vs-all)

Идея: построить классификаторы для каждой пары классов.

Получим $K(K-1)$ задач бинарной классификации:

- Объекты: $X^{km} = \{x \in X^l | y(x) = k \text{ или } y(x) = m\}$;
- Ответы: $y_i^{km} = [y_i = k]$;
- Оценка принадлежности: $b_{km}(x) \in R$.;
- Симметрия: $b_{km}(x) = -b_{km}(x)$.

Итоговый алгоритм:

$$a(x) = \operatorname*{argmax}_{k=1,\dots,K} \sum_{m=1}^{K} b_{km}(x)$$

Метрики качества

$$\frac{1}{l}\sum_{i=1}^{l}[a(x_i) = y_i]$$

Матрица ошибок:

	$y = 1$	…	$y = K$
$a(x) = 1$	q_{11}	…	q_{1K}
…	…	…	…
$a(x) = K$	q_{K1}	…	q_{KK}

$$q_{ij} = \sum_{m=1}^{l}[a(x_m) = i][y_m = j]$$

Метрики качества

Микро-усреднение (micro-averaging):

- Найдем TP, FP, FN, TN для каждой из задач;
- Усредним их по всем задачам;
- Вычислим итоговую метрику.

Вклад каждого класса зависит от его размера.

Макро-усреднение (macro-averaging):

- Вычислим итоговую метрику для каждой из задач;
- Усредним по всем классам.

Все классы вносят равный вклад.

Пример

	TP	FP	FN	TN
$y = 1$	900	120	100	930
$y = 2$	850	70	150	980
$y = 3$	10	100	40	1900

Микро-усреднение:

TP	FP	FN	TN
586.7	96.7	96.7	1270

Точность: 86%

Макро-усреднение:

Класс 1	Класс 2	Класс 3
88%	92%	9%

Точность: 63%

Правильная интерпретация значений решающей функции

```
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()
X = iris.data[:,:]
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3,
random_state=0)

clf = SVC(probability=True)
print(clf.fit(X_train,y_train).decision_function(X_test))
print(clf.predict(X_test))
print(y_test)
```

```
[[-0.76231668 -1.03439531 -1.40331645]
 [-1.18273287 -0.64851109  1.50296097]
 [ 1.10803774  1.05572833  0.12956269]
 [-0.47070432 -1.08920859 -1.4647051 ]
 [ 1.18767563  1.12670665  0.21993744]
 [-0.48277866 -0.98796232 -1.83186272]
 [ 1.25020033  1.13721691  0.15514536]
 [-1.07351583 -0.84997114  0.82303659]
 [-1.04709616 -0.85739411  0.64601611]
 [-1.23148923 -0.69072989  1.67459938]
 [-0.77524787 -1.00939817 -1.08441968]
 [-1.12212245 -0.82394879  1.11615504]
 [-1.14646662 -0.91238712  0.80454974]
 [-1.13632316 -0.8812114   0.80171542]
```

One-vs-one

Now that you already know OvR, OvA is not that much harder to grasp. You basically construct a classifier of every combination of class pairs (A, B). In your case: 0 vs 1, 0 vs 2, 1 vs 2.

Note: The values of (A, B) and (B, A) can be obtained from a single binary classifier. You only change what is considered the positive class and thus you have to invert the sign.

Doing this gives you a matrix:

A / B	#0	#1	#2
#0	--	-1.18	-0.64
#1	1.18	--	1.50
#2	0.64	-1.50	--

Read this as following: Decision function value when class A (row) competes against class B (column).

In order to extract a result a vote is performed. In the basic form you can imagine this as a single vote that each classifier can give: Yes or No. This could lead to draws, so we use the whole decision function values instead.

A / B	#0	#1	#2	SUM
#0	-	-1.18	-0.64	-1.82
#1	1.18	-	1.50	2.68
#2	0.64	-1.50	-	0.86

The resulting columns gives you again a vector `[-1.82, 2.68, 0.86]`. Now apply `arg max` and it matches your prediction.

One-vs-rest

I keep this section to avoid further confusion. The scikit-lear SVC classifier (libsvm) has a `decision_function_shape` parameter, which deceived me into thinking it was OvR (i am using liblinear most of the time).

For a real OvR respone you get one value from the decision function per classifier, e.g.

```
[-1.18273287 -0.64851109  1.50296097]
```

Now to obtain a prediction from this you could just apply `arg max`, which would return the last index with a value of `1.50296097`. From here on the decision function's value is not needed anymore (for this single prediction). That's why you noticed that your predictions are fine.

However you also specified `probability=True`, which uses the value of the distance_function and passes it to a sigmoid function. Sample principle as above, but now you also have confidence values (i prefer this term over probabilities, since it only describes the distance to the hyperplane) between 0 and 1.

```
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
import numpy as np

iris = datasets.load_iris()
X = iris.data[:,:]
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3,
random_state=0)

clf = SVC(decision_function_shape='ovo')  # EXPLICIT OVO-usage!
clf.fit(X, y)

def predict(dec):
    # OVO prediction-scheme
    # hardcoded for 3 classes!
    # OVO order assumption: 0 vs 1; 0 vs 2; 1 vs 2 (lexicographic!)
    # theory: http://www.stat.ucdavis.edu/~chohsieh/teaching/ECS289G_Fall2015/lecture9.pdf pa
    #    and: http://www.mit.edu/~9.520/spring09/Classes/multiclass.pdf page 8
    class0 = dec[0] + dec[1]
    class1 = -dec[0] + dec[2]
    class2 = -dec[1] - dec[2]
    return np.argmax([class0, class1, class2])

dec_vals = clf.decision_function(X_test)
pred_vals = clf.predict(X_test)
pred_vals_own = np.array([predict(x) for x in dec_vals])

for i in range(len(X_test)):
    print('decision_function vals  : ', dec_vals[i])
    print('sklearns prediction     : ', pred_vals[i])
    print('own prediction using dec: ', pred_vals_own[i])
```

```
decision_function vals  :  [-0.76867027 -1.04536032 -1.60216452]
sklearns prediction     :  2
own prediction using dec:  2
decision_function vals  :  [-1.19939987 -0.64932285  1.6951256 ]
sklearns prediction     :  1
own prediction using dec:  1
decision_function vals  :  [ 1.11946664  1.05573131  0.06261988]
sklearns prediction     :  0
own prediction using dec:  0
decision_function vals  :  [-0.46107656 -1.09842529 -1.50671611]
sklearns prediction     :  2
own prediction using dec:  2
decision_function vals  :  [ 1.2094164   1.12827802  0.1415261 ]
sklearns prediction     :  0
own prediction using dec:  0
decision_function vals  :  [-0.47736819 -0.99988924 -2.15027278]
sklearns prediction     :  2
own prediction using dec:  2
decision_function vals  :  [ 1.25467104  1.13814461  0.07643985]
sklearns prediction     :  0
own prediction using dec:  0
decision_function vals  :  [-1.07557745 -0.87436887  0.93179222]
sklearns prediction     :  1
own prediction using dec:  1
decision_function vals  :  [-1.05047139 -0.88027404  0.80181305]
sklearns prediction     :  1
own prediction using dec:  1
decision_function vals  :  [-1.24310627 -0.70058067  1.906847  ]
sklearns prediction     :  1
own prediction using dec:  1
decision_function vals  :  [-0.78440125 -1.00630434 -0.99963088]
sklearns prediction     :  2
own prediction using dec:  2
decision_function vals  :  [-1.12586024 -0.84193093  1.25542752]
sklearns prediction     :  1
own prediction using dec:  1
decision_function vals  :  [-1.15639222 -0.91555677  1.07438865]
```

Как получить из решающей функции оценку принадлежности к классу?

I took a look at the apis in sklearn.svm.* family. All below models, e.g.,

- sklearn.svm.SVC
- sklearn.svm.NuSVC
- sklearn.svm.SVR
- sklearn.svm.NuSVR

have a common interface that supplies a

```
probability: boolean, optional (default=False)
```

parameter to the model. If this parameter is set to True, libsvm will train a probability transformation model on top of the SVM's outputs based on idea of Platt Scaling. The form of transformation is similar to a logistic function as you pointed out, however two specific constants `A` and `B` are learned in a post-processing step. Also see this stackoverflow post for more details.

$$P(y=1|f) = \frac{1}{1+\exp(Af+B)}.$$

I actually don't know why this post-processing is not available for LinearSVC. Otherwise, you would just call `predict_proba(X)` to get the probability estimate.

Of course, if you just apply a naive logistic transform, it will not perform as well as a calibrated approach like Platt Scaling. If you can understand the underline algorithm of platt scaling, probably you can write your own or contribute to the scikit-learn svm family. :) Also feel free to use the above four SVM variations that support `predict_proba`.

Ссылки

- https://prateekvjoshi.com/2015/12/15/how-to-compute-confidence-measure-for-svm-classifiers/
- https://www.researchgate.net/post/How_do_I_calculate_Classification_Confidence_in_Classification_Algorithms_Supervised_Machine_Learning
- https://en.wikipedia.org/wiki/Platt_scaling

Ссылки

http://scikit-learn.org/stable/modules/multiclass.html

http://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html

http://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsOneClassifier.html