

Machine Learning and Large Scale Data Analysis

Assignment 5 Due: Thursday May 31, 2018 at 1:30 pm.

Please read the following two articles, we'll discuss them on Tuesday May 29.

<https://www.theguardian.com/commentisfree/2018/may/13/we-created-poverty-algorithms-wont-make-that-go-away><https://www.theguardian.com/uk-news/2018/may/15/uk-police-use-of-facial-recognition-technology-failure>1. *Multiclass boosting* (20 points)

In this problem we show how the reweighting for boosting is derived from a particular optimization problem. If g_m is the m 'th classifier, the weight update scheme for step m of boosting is given by

$$\begin{aligned} \tilde{w}_i &= \begin{cases} (C-1) \frac{1-e_m}{e_m} w_i & X_i \text{ misclassified by } g_m \\ w_i & \text{otherwise} \end{cases}, \\ w_i^{new} &= \frac{\tilde{w}_i}{\sum_i \tilde{w}_i}. \end{aligned} \tag{1}$$

Assume we have C classes and we code class c as:

$$Y = \mathcal{Y}_c = \underbrace{\left(-\frac{1}{C-1}, \dots, 1, \dots, -\frac{1}{C-1}\right)}_{\substack{\uparrow \\ c}},$$

and assume we create multiclass classifiers $g(X)$ that produce an output $g(X) = \mathcal{Y}_c$ if $g(X) = c$.

- (a) Show that if a point is correctly classified $Y^t g(X) = \frac{C}{C-1}$,
and if it is misclassified $Y^t g(X) = -C/(C-1)^2$.
- (b) Define the following loss on a collection g_1, \dots, g_m of m classifiers:

$$\begin{aligned} L(X, Y, \beta_1, \dots, \beta_m, g_1, \dots, g_m) = \\ \sum_{i=1}^n \exp \left[-\frac{1}{C} (\beta_1 Y_i^t g_1(X_i) + \beta_2 Y_i^t g_2(X_i) + \dots + \beta_m Y_i^t g_m(X_i)) \right] \end{aligned} \tag{2}$$

Assume we've already trained the classifiers g_1, \dots, g_{m-1} and we want to find the best 'next' classifier g_m . So we want to compute

$$\begin{aligned}\beta_m, g_m &= \arg \min_{\beta, g} \sum_{i=1}^n \exp \left[-\frac{1}{C} \left(\sum_{r=1}^{m-1} \beta_r Y_i^t g_r(X_i) + \beta_m Y_i^t g_m(X_i) \right) \right] \\ &= \arg \min_{\beta, T} \sum_{i=1}^n \tilde{u}_i \exp \left[-\frac{1}{C} (\beta_m Y_i^t g_m(X_i)) \right],\end{aligned}$$

where $\tilde{u}_i = \exp \left[-\frac{1}{C} (\sum_{r=1}^{m-1} \beta_r Y_i^t g_r(X_i)) \right]$. Denote

$$e_m(g) = \sum_{i=1}^n u_i \mathbf{1}[Y_i^t g(X_i) < 0].$$

Show that

$$g_m = \arg \min_g \sum_{i=1}^n \tilde{u}_i \mathbf{1}[Y_i^t g(X_i) < 0] = \arg \min_g \sum_{i=1}^n u_i \mathbf{1}[Y_i^t g(X_i) < 0] \doteq \arg \min e_m(g),$$

where $u_i = \tilde{u}_i / \sum_i \tilde{u}_i$. In other words g_m is chosen to minimize the weighted error rate, and show that

$$\beta_m = \frac{(C-1)^2}{C} \left[\log \left(\frac{1-e_m}{e_m} \right) + \log(C-1) \right].$$

where $e_m = \min_g e_m(g)$

Hint: For fixed β write the sum over the losses as

$$\sum_{i: Y_i^t g(X_i) > 0} w_i \exp[-\beta/(C-1)] + \sum_{i: Y_i^t g(X_i) < 0} w_i \exp[\beta/(C-1)^2].$$

(c) The new weights for the next classifier will now be

$$\tilde{u}_i^{new} = \tilde{u}_i \exp \left[-\frac{(C-1)^2}{C^2} \left(\log \frac{1-e}{e} + \log(C-1) \right) Y_i^t g_m(X_i) \right],$$

and $u_i = \tilde{u}_i^{new} / \sum_i \tilde{u}_i^{new}$.

Show inductively that $u_i = w_i$ defined in (1).

Thus the reweighting scheme for boosting is equivalent to minimizing the exponential loss in (2).

Note that we never find g_m that actually minimizes the weighted error loss, we find some classifier based on the weighted error for example by growing a tree.

2. Multiple trees on MNIST (50 points)

In python use the following two functions:

```
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
```

Using the mnist dataset split the data into 50,000 training, 10000 validation, 10000 test.

- (a) With the function `RandomForestClassifier` experiment with various stopping rules

`min_samples_split` - minimum number of samples in node to allow it to split.
`max_features` - number of random features from which to select the best, and
`n_estimators` - number of trees. Also use the option `criterion="entropy"`.

Setting up the classifier is done with this command:

```
clf = RandomForestClassifier(n_estimators=num_trees,  
min_samples_split=...,criterion="entropy",  
                           max_features=...)
```

Then you can call the functions `clf.fit`, `clf.predict`, and `clf.predict_proba` - which gives the output average probabilities.

Plot error rates on both training and validation data as a function of the number of trees.

Once you have found the best protocol check it on the test set.

- (b) The function `AdaBoostClassifier` implements SAMME, the boosting protocol described in problem 1. You define a classifier `clfb` (for example `RandomForestClassifier` with 1 tree), and pass it on to the function

```
AdaBoostClassifier(clfb,n_estimators=...,algorithm='SAMME')
```

Experiment with the same three parameters in this setting (stopping rule and number of splits in the definition of `clfb`) and number of trees. Compare the results to the random forest results in part (a).

- (c) Implement a different boosting reweighting rule, where $w_i = w_i * 1/e$, for each misclassified example. To do this you will need to grow trees one by one. The `clf.fit` function has a parameter `weights` which allows you to train the tree with your own weights on the data points. In this protocol you do not use the β 's, rather you aggregate the trees as in random forest by averaging the terminal probabilities of all the trees. Perform the same comparisons you did in (a),(b) with this protocol.
- (d) Finally try running an SVM with a Gaussian kernel on the same data and compare to the results with trees. Also compare the time to train and the time to test.

```
from sklearn.svm import SVC  
gsvm=SVC(kernel='rbf',C=1./lambda)
```

`lambda` - is the penalty on the square norm of the coefficients. The function takes the inverse of `lambda` as the parameter.

3. EM (30 points)

Let $X = (X_1, \dots, X_d)$ be a random vector where $X_j \in \{0, 1, 2\}$. Let Y be a random variable taking values in the set $\{1, \dots, d-3\}$. Write $P(Y = k) = \pi_k, k = 1, \dots, d-3$. Conditional on Y we assume the components of X are independent with the following distribution:

$$P(X_j = r | Y = k) = a_r, r = 0, 1, 2, \text{ for } j = k, k + 1.$$

$$P(X_j = r | Y = k) = b_r, r = 0, 1, 2, \text{ for } j = k + 2, k + 3.$$

$$P(X_j = r | Y = k) = c_r, r = 0, 1, 2, \text{ for all } j < k \text{ and } j > k + 3.$$

- (a) If a_2, b_1, c_0 are close to 1 describe what X would look like for different values of Y . Try to do it visually. What do the variables Y represent geometrically.
- (b) Write the log-joint distribution - $\log P(X_1, \dots, X_d, Y)$.
- (c) If you had a fully observed sample $X_i, Y_i, i = 1, \dots, n$ what would be the maximum likelihood estimates of $a, b, c, \pi_k, k = 1, \dots, d-3$. Remember that $a_2 = 1 - a_0 - a_1$. If you can't derive it analytically try to come up with an intuitive solution and explain.
- (d) Write pseudo code for the EM iterations to estimate these parameters if Y_i are unobserved. Important warning: In the E-step, when you compute the conditionals $P(Y_i = k | X_i; a, b, c, \pi)$, you shouldn't compute probabilities directly because these are products of many numbers less than 1 and become smaller than the roundoff error. Compute log-probabilities (using sums as in (b)) $v_{ik} = \log(P(Y_i = k, X_i; a, b, c, \pi))$, then use the function `scipy.misc.logsumexp` to get an efficient computation of the log of the denominator $d = \log \sum_k v_{ik}$. Then compute $\exp(v_{ik} - d)$.
- (e) 100 samples with $d = 20$ from such a distribution are found in the file 'X.npy'. Code the EM algorithm and provide estimates for a, b, c, π . To initialize the EM set $a_0 = b_0 = c_0 = .5, a_1 = b_1 = c_1 = .4, \pi_k = 1/(d-3)$.