Ensemble Methods

Project: Ensemble Methods and PAC-Bayesian Learning

Master MALIA

Guillaume Metzler
Institut de Communication (ICOM)
Université de Lyon, Université Lumière Lyon 2
Laboratoire ERIC UR 3083, Lyon, France
guillaume.metzler@univ-lyon2.fr

Abstract

The project has two parts. In the **first part**, you implement the ensemble methods of the course (bagging, boosting, stacking), compare them with each other and with simple baselines on at least fifteen datasets, with an evaluation metric adapted to each dataset, in order to understand *when* and *why* these methods help. In the **second part**, all the code is given: two PAC-Bayesian bounds and the algorithms that minimize them, for categorical distributions on a finite set of voters and Gaussian distributions on linear classifiers. You compare the two bounds, study what happens when the prior and the posterior change, and compare the test risk of ensembles learned in the classical way with that of the classifiers learned by minimizing a bound.

Rules and Expected work

Groups. 1 or 2 student(s).

Duration. The deadline is the **22nd of November** for the all the students except for the ones in a work-study program for who the deadline is the **22nd of December**.

Deliverables. A report of at most 15 pages in which you report your findings/analysis. You can add appendices if needed. Importance will be given to the reproducibility of your results (figure + tables) so the code shall also be submitted with a short `README` explaining how to reproduce each table and figure of your report. I suggest you to put everything on a Github account in case or to submit directly your work with all the needed stuff to run it.

Provided files. `code/benchkit.py` (datasets of the course: `real_dataset(name)`, labels in $\{-1, +1\}$), `code/pacbayes_tools.py` (Part 2, complete code), `common/` and `data/`.

Tools. `numpy`, `scipy`, `scikit-learn`, `matplotlib`.

Evaluation. Given the context of this course, I will give more importance to the first part, approximately 60% to 75% of the final grade.

1 Ensemble methods: implementation and comparison

Methods.

- **Implement yourselves**: **bagging** of decision trees, with its out-of-bag error; **AdaBoost** (discrete, labels in $\{-1, +1\}$, base trees of a given depth); **stacking** (base models of different families, a meta-model trained on *out-of-fold* predictions). Check each implementation against its `scikit-learn` counterpart.
- Also use: random forests, gradient boosting (`GradientBoostingClassifier` and `HistGradientBoostingClassifier`).
- **Baselines**: a single decision tree, and at least one model of another family (logistic regression, k nearest neighbours...).

Datasets. At least **15 datasets** among the 25 of the course (`bk.REAL_SMALL`, `bk.REAL_LARGE`). Describe them in a table: number of examples m, dimension d, proportion of the minority class. Some of them are strongly **imbalanced** (`yeast3`, `libras`, `satimage`, `pageblocks`, `abalone8`...).

Evaluation metric. Choose the metric according to the nature of each dataset, and justify it.

- On a **balanced** dataset, the accuracy (or the error rate) is a natural choice; the AUC is a useful complement.
- On an **imbalanced** dataset, the accuracy is misleading: with 10% of positives, always answering "negative" gives 90% of accuracy. Use metrics that give the same importance to both classes or focus on the minority class: **balanced accuracy** (mean of the recalls of the two classes), **F1-score of the minority class**, **AUC**. State the threshold you use to call a dataset imbalanced.
- Show, on the imbalanced datasets, how the ranking of the methods changes with the metric.

Analysis. Results tables and or figures, average ranks, a possible statistical test to emphasize the significance of a given method. Your conclusions should answer at least:

- How much do the ensemble methods gain over a single tree? Over a model of another family?
- Bagging or boosting: on which datasets does each one win, and why (size of the dataset, noise, depth of the base trees, bias and variance)?
- Does stacking beat the best of its members? At what cost?
- On imbalanced datasets: which methods suffer, and what changes when the metric changes?

Report. For this first part: I expect a brief reminder of the used algorithms and a complete description of the followed experimental protocol. The results section will be based on the answers of the above questions and additional comments you may thnk relevant.

2 PAC-Bayesian bounds and their minimization (code given)

Reminders A distribution Q (posterior) on a set $\mathcal{H}$ of classifiers defines the **Gibbs classifier** G_Q (draw $h \sim Q$, predict $h(x)$) and the **majority vote** $B_Q(x) = \operatorname{sign}\left(\mathbb{E}_{h\sim Q} h(x)\right)$, with $R_{\mathcal{D}}(B_Q) \leq 2R_{\mathcal{D}}(G_Q)$. For a prior P fixed *before* seeing the sample S of size m, each of the following bounds holds with probability at least $1-\delta$, for all Q:

$$\text{(A, linear, Catoni)} \quad R_{\mathcal{D}}(G_Q) \leq \widehat{R}_S(G_Q) + \frac{\mathrm{KL}(Q\|P) + \ln\frac{|\Lambda|}{\delta}}{\lambda} + \frac{\lambda}{8m} \quad \text{for all } \lambda \in \Lambda,$$

$$\text{(B, McAllester)} \quad R_{\mathcal{D}}(G_Q) \leq \widehat{R}_S(G_Q) + \sqrt{\frac{\mathrm{KL}(Q\|P) + \ln\frac{2\sqrt{m}}{\delta}}{2m}},$$

$$\text{(Seeger)} \quad R_{\mathcal{D}}(G_Q) \leq \overline{\mathrm{kl}}^{-1}\Big(\widehat{R}_S(G_Q),\ \frac{\mathrm{KL}(Q\|P) + \ln\frac{2\sqrt{m}}{\delta}}{m}\Big),$$

where Λ is a grid of values of λ fixed in advance.

Available code. **All the code** is in `code/pacbayes_tools.py`: the three bounds, and for each of bounds A and B the algorithm that minimizes it, in two settings. Your work is to **run it, compare, and explain**.

	categorical Q, P on decision stumps	**Gaussian** Q, P on linear classifiers
voters	stumps $\pm\operatorname{sign}(x_j - \theta)$ (prior part)	$x \mapsto \operatorname{sign}\langle \mathbf{v}, x\rangle$, $\mathbf{v} \in \mathbb{R}^{d+1}$
prior / posterior	weights $P(h)$, $Q(h)$	$\mathcal{N}(\mathbf{w}_0, \sigma^2 I)$, $\mathcal{N}(\mathbf{w}, \sigma^2 I)$
$\mathrm{KL}(Q\|P)$	$\sum_h Q(h) \ln \frac{Q(h)}{P(h)}$	$\|\mathbf{w} - \mathbf{w}_0\|^2/(2\sigma^2)$
Gibbs risk / vote	$\sum_h Q(h)\widehat{R}_S(h)$ / weighted vote	probit loss / $\operatorname{sign}\langle \mathbf{w}, x\rangle$
bound A	Gibbs posterior $Q \propto Pe^{-\lambda \widehat{R}_S}$ for each λ	L-BFGS for each λ
bound B	L-BFGS on $Q = \operatorname{softmax}(\theta)$	L-BFGS on $\mathbf{w}$

Protocol (functions `split` and `standardize`): each dataset is split into a *prior part* (14%: thresholds of the stumps, standardization, possibly a learned prior), the *bound sample* S (56%: learning Q and computing the certificate) and a *test set* (30%: only to measure the true risks at the end). The prior never sees S. The functions `run_categorical` and `run_gaussian` do everything for one dataset and one split.

The code

The bounds:

```
54 def bound_linear(emp, KL, m, lam, delta=DELTA, n_grid=len(GRID)):
55     """Bound A (Catoni), valid for all lam of a grid of size n_grid (union bound: delta / n_grid)."""
56     return emp + (KL + np.log(n_grid / delta)) / lam + lam / (8 * m)
57
58
59 def bound_mcallester(emp, KL, m, delta=DELTA):
60     """Bound B (McAllester)."""
61     return emp + np.sqrt((KL + np.log(2 * np.sqrt(m) / delta)) / (2 * m))
62
63
64 def bound_seeger(emp, KL, m, delta=DELTA):
65     """Seeger's bound (kl version): the tightest of the three."""
66     return kl_inv_upper(emp, (KL + np.log(2 * np.sqrt(m) / delta)) / m)
```

Minimizing bound A and bound B on stumps (categorical distributions):

```python
def gibbs_posterior(P, L, lam):
    """Q(h) proportional to P(h) exp(-lam L(h)): the minimizer of  Q.L + KL(Q||P) / lam."""
    w = np.log(P) - lam * (L - L.min())
    Q = np.exp(w - w.max())
    return Q / Q.sum()


def learn_linear_categorical(P, L, m, grid=GRID, delta=DELTA):
    """Bound A: Gibbs posterior for each lam of the grid; keep the one with the smallest bound A."""
    rows = []
    for lam in grid:
        Q = gibbs_posterior(P, L, lam)
        emp, KL = float(Q @ L), kl_categorical(Q, P)
        rows.append(dict(lam=lam, Q=Q, emp=emp, KL=KL, bound=bound_linear(emp, KL, m, lam, delta, len(grid)),
                         seeger=bound_seeger(emp, KL, m, delta)))
    return min(rows, key=lambda r: r["bound"]), rows


def learn_mcallester_categorical(P, L, m, delta=DELTA):
    """Bound B: minimize Q.L + sqrt((KL(Q||P) + c) / (2m)) over Q = softmax(theta), by L-BFGS."""
    c = np.log(2 * np.sqrt(m) / delta)

    def f(theta):
        Q = np.exp(theta - theta.max()); Q /= Q.sum()
        KL = kl_categorical(Q, P)
        r = np.sqrt((KL + c) / (2 * m))
        gQ = L + (np.log(np.maximum(Q, 1e-300) / P) + 1) / (4 * m * r)      # gradient w.r.t. Q
        return float(Q @ L) + r, Q * (gQ - Q @ gQ)                            # chain rule through softmax

    theta = minimize(f, np.log(P), jac=True, method="L-BFGS-B").x
    Q = np.exp(theta - theta.max()); Q /= Q.sum()
    emp, KL = float(Q @ L), kl_categorical(Q, P)
    return dict(Q=Q, emp=emp, KL=KL, bound=bound_mcallester(emp, KL, m, delta), seeger=bound_seeger(emp, KL, m, delta))
```

Minimizing bound A and bound B on linear classifiers (Gaussian distributions):

```python
def learn_linear_gaussian(X, y, w0, sigma=1.0, grid=GRID, delta=DELTA):
    """Bound A: for each lam, minimize probit_risk(w) + KL / lam (L-BFGS, warm start); keep the best bound A."""
    m, A = len(y), _probit_parts(X, y, sigma)

    def f(w, lam):
        t = A @ w
        return (np.mean(norm.cdf(-t)) + np.sum((w - w0) ** 2) / (2 * sigma ** 2 * lam),
                -(norm.pdf(t) @ A) / m + (w - w0) / (sigma ** 2 * lam))

    rows, w = [], w0.copy()
    for lam in grid:
        w = minimize(f, w, args=(lam,), jac=True, method="L-BFGS-B").x
        emp, KL = probit_risk(w, X, y, sigma), kl_gaussian(w, w0, sigma)
        rows.append(dict(lam=lam, w=w.copy(), emp=emp, KL=KL, bound=bound_linear(emp, KL, m, lam, delta, len(grid)),
                         seeger=bound_seeger(emp, KL, m, delta)))
    return min(rows, key=lambda r: r["bound"]), rows


def learn_mcallester_gaussian(X, y, w0, sigma=1.0, delta=DELTA):
    """Bound B: minimize probit_risk(w) + sqrt((KL + c) / (2m)) directly (L-BFGS)."""
    m, A = len(y), _probit_parts(X, y, sigma)
    c = np.log(2 * np.sqrt(m) / delta)

    def f(w):
        t = A @ w
        r = np.sqrt((np.sum((w - w0) ** 2) / (2 * sigma ** 2) + c) / (2 * m))
        return (np.mean(norm.cdf(-t)) + r,
                -(norm.pdf(t) @ A) / m + (w - w0) / (sigma ** 2 * 4 * m * r))

    w = minimize(f, w0 + 1e-3, jac=True, method="L-BFGS-B").x
    emp, KL = probit_risk(w, X, y, sigma), kl_gaussian(w, w0, sigma)
```

```
    return dict(w=w, emp=emp, KL=KL, bound=bound_mcallester(emp, KL, m, delta), seeger=bound_seeger(
     emp, KL, m, delta))
```

The whole pipeline for one dataset (the Gaussian version `run_gaussian` is similar):

```
def run_categorical(X, y, seed=0, prior="uniform"):
    """Both bounds on decision stumps. prior = 'uniform' or 'learned' (Gibbs posterior on the prior
     part)."""
    Xp, yp, Xs, ys, Xte, yte = split(X, y, seed)
    H = make_stumps(Xp)
    V, Vte = stump_votes(H, Xs), stump_votes(H, Xte)
    P = np.full(len(H), 1 / len(H))
    if prior == "learned":
        P = np.maximum(gibbs_posterior(P, voter_risks(stump_votes(H, Xp), yp), len(yp) / 4), 1e-300)
        P /= P.sum()
    L, m = voter_risks(V, ys), len(ys)
    A, _ = learn_linear_categorical(P, L, m)
    B = learn_mcallester_categorical(P, L, m)
    out = {}
    for name, r in [("A (linear)", A), ("B (McAllester)", B)]:
        out[name] = dict(bound=r["bound"], seeger=r["seeger"], KL=r["KL"], emp=r["emp"], neff=1 / np.
     sum(r["Q"] ** 2),
                         test_gibbs=gibbs_risk(r["Q"], Vte, yte), test_vote=vote_risk(r["Q"], Vte,
     yte))
    return out
```

What you have to do

1. **Compute the bounds.** Using the three functions `bound_*`, plot the bounds as functions of m (for $\widehat{R}_S = 0.1$ and a few values of the KL) and of the KL (for a few m). Which bound is the tightest? How does bound A depend on λ, and what is the price of the grid? Relate what you see to the formulas.

2. **Compare the two bounds.** On `wdbc`, then on at least five other datasets and several splits, compare the two learning algorithms (`run_categorical`): value of the bound they minimize, Seeger's certificate of the posterior they return, KL, effective number of voters $1/\sum_h Q(h)^2$, test risk of the Gibbs classifier and of the vote. Do the two bounds lead to the same posterior? Which certificate would you report, and why? For bound A, plot the bound, the KL and the test risks against λ (second output of `learn_linear_categorical`).

3. **Impact of the prior.** Categorical: uniform prior versus a prior learned on the prior part (`prior="learned"`). Gaussian: $\mathcal{N}(\mathbf{0}, I)$ versus $\mathcal{N}(\mathbf{w}_0, I)$ with $\mathbf{w}_0$ learned on the prior part. What changes in the KL, the certificates and the test risks? Why is a learned prior legitimate here?

4. **Changing the family of distributions.** Categorical distributions on stumps versus Gaussian distributions on linear classifiers: compare certificates, Gibbs risks and votes. You may add other posteriors (a Dirac on the best stump; uniform on the k best stumps) with the functions given. Why does minimizing a bound on the Gibbs risk give a weak vote of stumps, but a good linear classifier?

5. **Classical learning versus bound minimization.** On at least six datasets and several splits, compare the test risks of ensembles learned in the classical way on the whole training part (AdaBoost on stumps, random forest, your best method of Part 1, logistic regression) with those of the classifiers obtained by minimizing the bounds, and give the certificates. Which approach predicts better? What does the PAC-Bayesian approach offer that the classical one does not, and at what price?

Report. For the second part: the figures and tables of questions 1 to 5, each followed by its interpretation, and a short conclusion on what the PAC-Bayesian approach brings to ensemble methods.

Bibliography

J. Demšar. Statistical comparisons of classifiers over multiple data sets. *JMLR*, 7, 2006.

P. Germain, A. Lacasse, F. Laviolette, M. Marchand. PAC-Bayesian learning of linear classifiers. *ICML*, 2009.

H. He, E. A. Garcia. Learning from imbalanced data. *IEEE TKDE*, 21, 2009.

J. Langford, J. Shawe-Taylor. PAC-Bayes & margins. *NIPS*, 2002.

D. McAllester. PAC-Bayesian model averaging. *COLT*, 1999.