

Voting Assemblies: Adaboost

Jerry Bonnell, Gururaj Shriram, and Lloyd Beaufils

ECE548 Group 7

November 29, 2017

Why bother **boosting**?

- Data used for induction rarely captures all aspects of the class.
- Individual classifiers suffer from **bias-related** error.
- How about using **many** classifiers? “Collective wisdom”?

A bit of theory...

Adaboost

“Examples are chosen from a probabilistic distribution that is gradually modified in a way that makes the whole “assembly” focus on those aspects that appear to be difficult.”

- Miroslav Kubat, *An Introduction to Machine Learning*

- Improvement over **bagging** and **Schapire's boosting**.
 - Reduce randomness and the need for great many examples.
- Classifiers should build upon the weaknesses of others.

A Practical Approach

- Selecting examples for T_i is done **probabilistically**.
- For T_1 : each example has $p = 1/m$
 - "Wheel of Fortune"
- For each following T_i :
 - Calculate $\varepsilon_i = \sum_{j=1}^m p_i(\mathbf{x}_j) * e_i(\mathbf{x}_j)$
 - Calculate $\beta_i = \varepsilon_i / (1 - \varepsilon_i)$
- Scale **correctly** classified examples by $p_{i+1}(\mathbf{x}_j) = p_i(\mathbf{x}_j) * \beta_i$
 - Less chance of being selected.
- Normalize to ensure the new distribution sums to 1.
- From each "custom" T_i , a C_i is induced.

Weighted Majority Voting

- Each classifier's vote has a weight.
- Higher classifier performance = votes worth more = higher weight
- Sum these weights for W_{pos} and W_{neg} to see which has higher support.
- How to obtain these weights? Use **perceptron learning**.
 - Present the assembly one example at a time from the training set.
 - Upon misclassification, use classical weight-updating formula (Chapter 4)

Hypothesis

- Training and testing error rate should **decrease** when more classifiers added.
 - We would like to see weak learners approximate the accuracy of a strong learner.
- Error rate should **converge** after some large number of classifiers.
- Computational costs → reap benefits of using small training subsets.

Does experience agree?

Implementation

- Weak learner: **Perceptron** (ours, in Python)
- Strong learner: **Decision Tree** (Scikit)
- Learning rate $\eta = 0.05$ and $\eta_{\text{weights}} = 0.0001$
- Threshold $\theta = 0.05$ and $\theta_{\text{weights}} = 0.10$
- Each training subset T_i will hold 20% of the training set.
- Training set 65% / Testing set 35%

Experiment

- Begin with one learner. Add groups of two till 50 reached.
- Average this process for 5 times.
- Measure **# classifiers** vs. **error rate**

Domains

UCI Repository

- 1 **Musk** (168 attributes, 476 instances, 47 pos / 45 neg)
- 2 **Ionosphere** (34 attributes, 351 instances, 225 pos / 126 neg)
- 3 **insert** (xx attributes, xx instances, xx pos / xx neg)

- Exclusively **binary** classification tasks.

Musk

Insert graph here

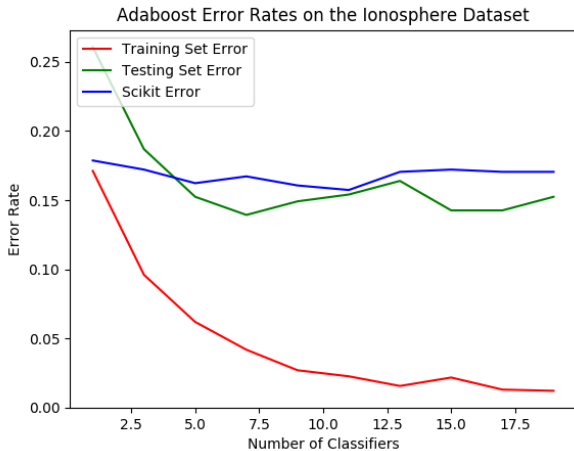
Musk

Insert another graph here

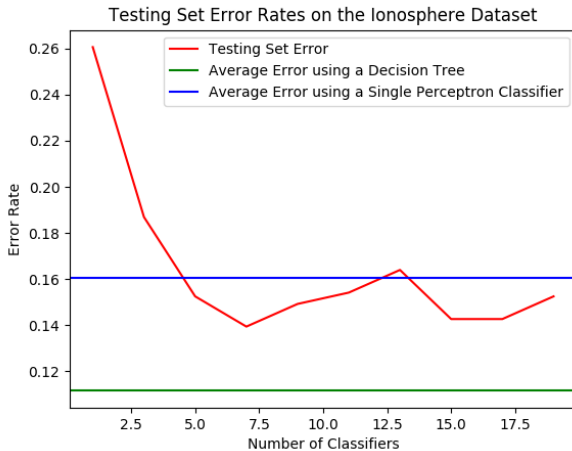
Musk Observations

Insert another graph here

Ionosphere



Ionosphere



Ionosphere Observations

- observation

Insert

graph here

Insert

graph here

Insert Observations

- observation

Conclusions

- concluding remarks

Questions?