Voting Assemblies: Adaboost

Jerry Bonnell, Gururaj Shriram, and Lloyd Beaufils

ECE548 Group 7

December 4, 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 $\mathrm{T_i}$ is done **probabilistically**.
- For $\mathrm{T_1}$: each example has $p = 1/m$
  - "Wheel of Fortune"
- For each following T_i:
  - Calculate $\varepsilon_\mathrm{i} = \sum_{j=1}^{m} p_\mathrm{i}(\mathbf{x}_\mathrm{j}) * e_\mathrm{i}(\mathbf{x}_\mathrm{j})$
  - Calculate $\beta_\mathrm{i} = \varepsilon_\mathrm{i}/(1 - \varepsilon_\mathrm{i})$
- Scale **correctly** classified examples by $p_{\mathrm{i}+1}(\mathbf{x}_\mathrm{j}) = p_\mathrm{i}(\mathbf{x}_\mathrm{j}) * \beta_\mathrm{i}$
  - ε_i small $\rightarrow$ reduce chance of selection.
- Normalize to ensure the new distribution sums to 1.
- From each "custom" $\mathrm{T_i}$, a $\mathrm{C_i}$ 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.
  - Minimize **variance-based** error.
- We would like to see weak learners approximate the accuracy of a strong learner.
- Error rates should **plateau** after some large number of classifiers.

Does experience agree?

Implementation

- Adaboost in Python
- 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 $\theta_{\text{weights}} = 0.05$
- Each training subset T_i will hold 20% of the training set.
- Training set 65% / Testing set 35%

Experiment

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

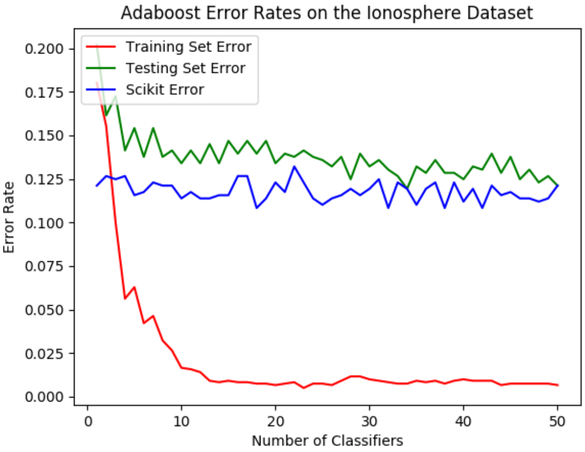
Domains

UCI Repository

1. **Ionosphere** (34 attributes, 351 instances, 64.1% pos / 35.9% neg)
2. **Breast Cancer** (10 attributes, 683 instances, 34.9% pos / 65.1% neg)
3. **Musk** (168 attributes, 476 instances, 43.4% pos / 56.6% neg)

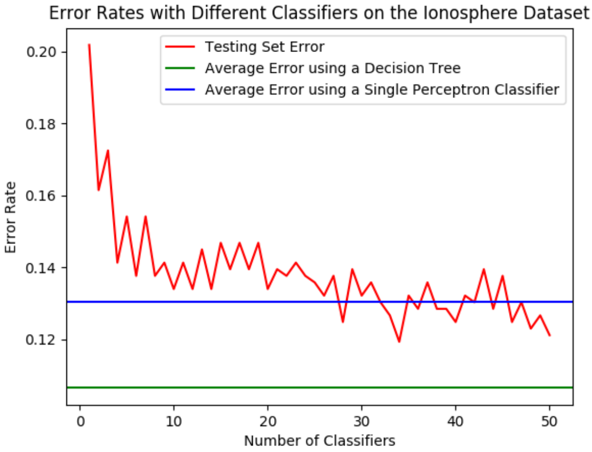
- Exclusively **binary** classification tasks.

Ionosphere



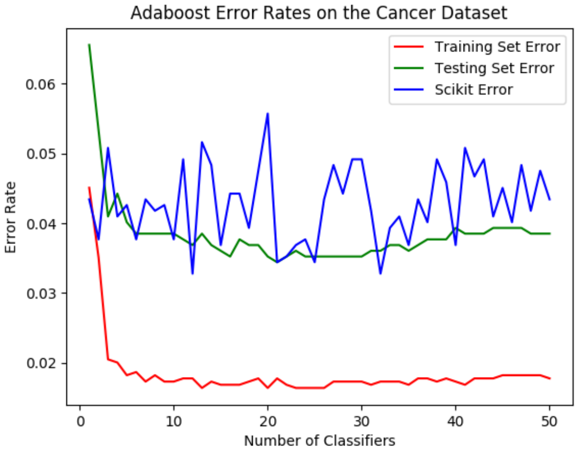
Adaboost Error Rates on the Ionosphere Dataset
Training Set Error
Testing Set Error
Scikit Error
Error Rate
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
10
20
30
40
50
Number of Classifiers

Ionosphere

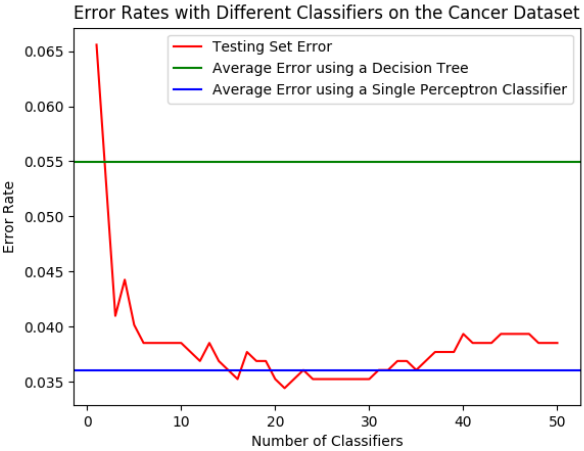


Error Rates with Different Classifiers on the Ionosphere Dataset
Testing Set Error
Average Error using a Decision Tree
Average Error using a Single Perceptron Classifier
0.20
0.18
0.16
0.14
0.12
Error Rate
0
10
20
30
40
50
Number of Classifiers

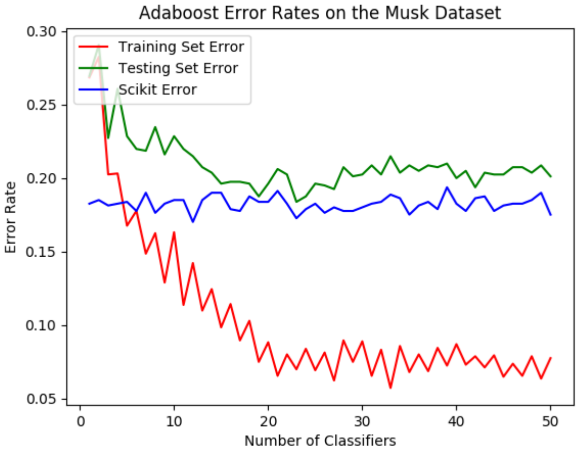
Breast Cancer



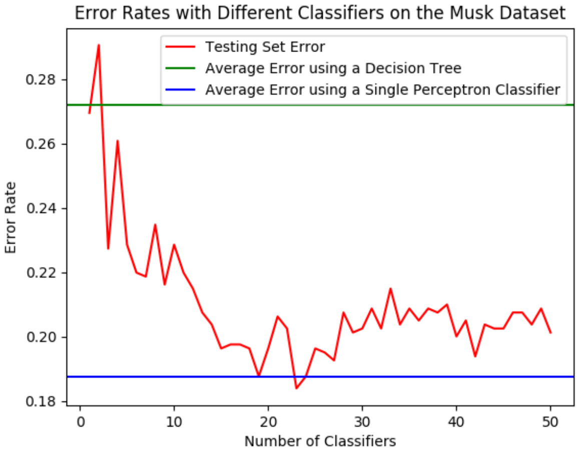
Breast Cancer



Musk


Adaboost Error Rates on the Musk Dataset
Training Set Error
Testing Set Error
Scikit Error
Error Rate
0.30
0.25
0.20
0.15
0.10
0.05
0
10
20
30
40
50
Number of Classifiers

Musk



Error Rates with Different Classifiers on the Musk Dataset
Testing Set Error
Average Error using a Decision Tree
Average Error using a Single Perceptron Classifier
Error Rate
0.18
0.20
0.22
0.24
0.26
0.28
0
10
20
30
40
50
Number of Classifiers

Conclusions

- Adaboost drives the error rate down on testing set with increasing number of classifiers.
- Shows promise in situations with **imbalanced** training sets.
- However, little "boosting" observed in more equally represented datasets (Musk)
- Would have liked to test with 100+ classifiers to observe greater convergence in error.

Questions?