Ensemble Classifiers - Paper Review #1

Matt Triano

1: Main Components of Ensemble Learning Frameworks

Individual learning models can be tuned to be fairly precise, but a single learner is akin to a single data point. It could be brilliant, but you can't get a sense for the learner's accuracy without a distribution of predictions. Further, it's difficult to know if that learner just studied for the test, or if they have deep knowledge that can generalize beyond the data at hand. And even further, how can we be sure that the learner is the best possible learner for all scenarios? To address these concerns, we can assemble an ensemble of (potentially) similar learners and treat them as a distribution.

This strategy is generalized into ensemble learning frameworks, which typically involve these standard components:

- **Training Data** that contains many observations containing labeled values and 1 to many corresponding attributes.
- A **base induction algorithm** or model the captures the general relationship between the labeled values and the corresponding attributes.
- A **"diversity generator"** that takes the base induction algorithm (described above) and generates many new classifiers of that form but tweaked to so that they're not all the same classifier. This can be done either by training them with a different subset of the data, changing internal weightings, or other ways.
- An **aggregator** that combines the predictions of these diverse classifiers.

By polling many classifiers, the weaknesses or strengths of any specific classifier have less pull on the overall trend (ie we can reduce both bias and variance) and we can start calculating the stability of the predictions.

2: AdaBoost Variations for *Big Data*

In general, **AdaBoost** is an iterative process where it builds a new classifier based on the previous classifier each iteration. If the system has to do n_c operations per classification and iterates n_i times, then the process has a complexity of $\mathcal{O}(n_i n_c)$. Building a decision tree classifier, for example, takes $(mn \log_2 n)$ time (for a tree with m features and n data samples), which is significantly slower than linear time ($\mathcal{O}(n)$). And in general, even the fastest algorithms (like Naive Bayes) would still have to run through every datgpoint every iteration. A cursory examination of these concepts indicates a problem: increasing the number of datapoints can have an exponential increase on the the runtime.

Huge datasets can contain extremely valuable information, and AdaBoost is good a building strong learners, but an algorithm isn't very useful if it takes too long to run. To speed up AdaBoost, earlier data scientists found ways to make AdaBoost more time-efficient in these adaptations:

- **P-AdaBoost** takes advantage of the fact that, on average, the marginal model error drops fastest (in both absolute and relative terms) in the first n_i iterations than in iterations beyond n_i, so P-AdaBoost only runs in standard sequential fashion for a limited number of steps. Then, it uses that most recent model as the base inductor and builds many diversified classifiers. Then, those classifiers can be run in parallel (possibly on many different machines) and the results can be aggregated. This allows us to use hardware and electricity in stead of time.
- Whene there is generally low noise, **Local Boosting** can converge on ideal weights faster than AdaBoost. On each iteration, AdaBoost calculates a global error rate to apply to all of the separate weak learners, and this can allow signicant outliers to have disporoportional impact on the weights calculated for learners. To address that, local boosting inspects error rates in similar learner instances rather than in all learner instances. If many similar learner instances are correctly classified, then the misclassified learned in that group are more likely to be noisy data, and their weight can be decreased. If the majority of the similar points are misclassified, then that signals that there is variance here that the model isn't capturing, and it can increase the weight of these points. If the amount of noise is low, **Local Boosting** can outperform the accuracy of Random Forest or Bagging, but as Local Boosting has to keep track of local subsets, it increases the data storage requirements.
- Another AdaBoost Variant that seeks to improve accuracy by exploiting error rate information is **BoostMA**, which increases a learner's weighting if they performed worse than default. This highlights the information that the aggregate model was failing to capture on the previous iteration (which resulted in the misclassification.

3: Combining Base-Classifier Outputs:

There are a rich array of methods for combining the predictions of different learners.

Weighting Methods

Combination strategies based on learner weights assign influence (in some way) proportional to their weight.

- **Majority Voting** is very simple combination strategy where the predictions of all learners are given equal weight. The aggregate prediction is the one that received the most votes. It is conceptually very simple, so it is typically used in initial explorations.
- **Performance Weighting** sets the weight of each learner proportional to its accuracy in predicting the values in the known test set.
- **Vogging** (or **V**ariance **O**ptimized **B**agging) adjusts weights so that the aggregate linear combination of all learners minimizes variance, ie produces the most consistent response (whether correct or not). In practice, a minimum accuracy value is set to prevent the algorithm from simply optimizing for variance at the complete expence of accuracy.

Meta-Learner Methods

Meta-learning is useful if classifier models can consistently correctly classify (or misclassify) certain classes of data.

- **Stacking** seeks to identify the classifiers that are reliable. It does this by making a new dataset using the actual labeled data and using the results of the ensemble of classifiers as the corresponding attributes. With this new dataset, a holdout portion can be separated (or you can jump straight to cross validation), and then the training set of this new dataset is used to build a meta-classifier. You basically stack another classifier on top of the aggregation of earlier classifiers. This method generalizes very well to new data.
- **Grading** looks at the classification made by each pair of classifier and training set and records if that pairing produces a correct prediction or an incorrect prediction, without taking class or confusion matrix results (e.g. false positive, true positive, false negative, true negative) into account. Then, for each classifier, the grader trains a metaclassifier from that classifier's classifications over the range of different training sets, and tries to predict when a classifier will predict correctly. Then the classifiers are given unlabeled observations to classify and the metaclassifier predicts what the classifier will predict. Ultimately, the proceedure settles on a subset of classifiers that were metaclassified most accurately. This has the effect of identifying the most predictable and accurate classifiers and filtering out the rest.

4: Diversity's Bounty

As expressed earlier, in ensembles of classifiers, 'diversity' is the quality of being similar to a base induction algorithm, but not identical. This is similar to diversity in humans, who are mostly generated through a similar process from a nearly identical blueprint, undergo a long period of training with a specific training set, and then try to generalize that training to new data; mostly similar, but the differences in training data can lead to different predictions. Whereas people/classifiers that were trained with training data set D_{rural} may excel at things like animal identification or plant cultivation but they may struggle with things that the people/classifiers trained with the D_{urban} training data set excel at, like navigate public transit, or appropriately handle requests from strangers. This enjoyable analogy (at least I enjoyed writing it) describes how **manipulating/varying the training samples** can produce diversity.

Another way to produce diversity around generally similar classifiers (or people) is to **partition the search space**. Stated less abstractly, the available data could be partitioned or clustered and then classifiers could be started in these separated data regions and then begin exploring and training themselves. After a training period, new data can be introduced to the partitioned clusters (assuming it met the criteria that generated that subset) and this process can evaluate the classifiers.