CS60050 Machine Learning - Weekly Report

Nisarg Upadhyaya (19CS30031)

Week 11: 28-29 October

1 Topics covered

- No Free Lunch Theorem
- Generating Diverse Learners
- Model Combination Schemes
- Voting
- Error-Correcting Output Codes
- Bagging
- Boosting
- Mixture of Experts
- Stacked Generalisation
- Cascading

2 Summary

2.1 No Free Lunch Theorem

There is no single learning algorithm that in any domain always induces the most accurate learner. The usual approach is to try many and choose the one that performs the best on a separate validation set. The inductive bias leads of an algorithm to error if the assumptions do not hold for the data. By suitably combining multiple base learners we can improve accuracy. This leads to increase in time and space complexity.

2.2 Generating Diverse Learners

We need to maximise individual accuracies while having diversity between learners. It can be achieved in the following ways:

1. Using different algorithms to train different base learners.
2. Using the same learning algorithm but with different hyperparameters.
3. Using different input representations by integrating different types of measurments ans data from different sensors. We can also choose random subsets of the input features, this way different learners look at the data from different views and also it helps reduce dimensions.
4. Using different training sets by either drawing randomly which is called bagging or training serially where instances on which the preceding base-learners are not accurate are given more emphasis which includes boosting and cascading.

2.2.1 Diversity vs Accuracy

When we have different base learners, we want them to be reasonably accurate but not very accurate individually. The base-learners are not chosen for their accuracy, but for their simplicity. We require that the base learners be diverse and accurate on different instances, specializing in sub domains of the problem. The final accuracy when the base-learners are combined should be good.

2.3 Model Combination Schemes

We can have multiexpert combination where base-learners work in parallel. In global approach output from all learners is used. In local approach selected learners are used to get the output. We can also have multistage combination where the next base-learner is trained with or tested on only the instances where the previous base-learners are not accurate enough. If we denote by $d_j(x)$ the prediction of the j^{th} base learner the final prediction is calculated as $y = f(d_1, d_2, ..., d_L|\Phi)$ where $f(\cdot)$ is the combining function with Φ denoting its parameters.

2.4 Voting

In this we take a linear combination of the learners, $y_i = \sum_j w_j d_{ji}$ such that $w_j \geq 0, \sum_j w_j = 1$. This is one way. There are other ways in we can combine these functions such as average, max, min, median, product etc. Voting schemes can be seen as approximations under a Bayesian framework. In classification we have $w_j = P(M_j)$ and $d_{ji} = P(C_i|x, M_j)$ and thus we have $P(C_i|x) = \sum_{M_j} P(C_i|x, M_j)P(M_j)$.

2.5 Error-Correcting Output Codes

In this we have for each class a set of binary classification tasks predefined. Base-learners are binary classifiers having output $-1/+1$, and there is a code matrix W of size $K \times L$ whose K rows are the binary codes of classes in terms of the L base-learners. The columns define the partitioning of the classes into two sets labelled either $+1$ or -1. The codes belonging to a class should follow error correcting codes principle. Approach is to increase the Hamming distance. Thus we would like the columns of W to be as different as possible so that the tasks to be learned by the base-learners are as different from each other as possible.

2.6 Bagging

This is a voting method whereby each base-learner is trained over slightly different training sets, sampled from the original training set with replacement. Thus it is possible some instances are drawn more than once while some are not drawn at all. Bagging uses bootstrap to generate L training sets, trains L base-learners using an unstable learning procedure, and then, during testing, takes an average.

2.7 Boosting

In boosting, we actively try to generate complementary base-learners by training the next learner on the mistakes of the previous learners. A week learner has probability less than $\frac{1}{2}$ and a strong learner has arbitrarily small error probability.

1. Original algorithm (using three weak learners): Randomly divide training set into three. Use X_1 to train d_1. Take X_2 and feed it to d_1 and take all instances misclassified by d_1 and also as many instances on which d_1 is correct from X_2 to form the training set of d_2. Take X_3 and feed it to d_1 and d_2. The instances on which d_1 and d_2 disagree form the training set of d_3. During testing feed the instance to d_1 and d_2, if they agree, that is the output, otherwise the output of d_3 is taken as the output.
2. AdaBoost (adaptive boosting): It uses the same training set over and over, thus it need not be large. The classifiers should be simple however. At each iteration i train with the sample set and compute the training error ϵ of classification. If $\epsilon > \frac{1}{2}$ we stop. Otherwise include the model d_i in the list and update the sampling probability of each t^{th} training sample and normalise these new probabilities. Once training is done, AdaBoost is a voting method. Given an instance, all d_j decide and a weighted vote is taken where weights are proportional to the base-learner's accuracies on the training set.

2.8 Mixture of Experts, Stacked Generalisation and Cascading

1. Mixture of experts: While in voting the weights are fixed for each expert, here we have a gating network whose outputs are the weights of the experts. Hence this can be seen as a voting method where votes vary on different inputs. Experts in the locality of the input get weights close to 1 and the others close to 0.
2. Stacked generalisation: It extends voting in the way that the output instead of just a linear combination of base learners, can be any general functional form with parameters Φ which are also learned. Thus we have $y = f(d_1, d_2, ..., d_L|\Phi)$. It can be implemented as a multilayer perceptron.
3. Cascading: In cascading we order the base learners in terms of their complexity with less complex models first. Each learner produces an output with a certain confidence. The next learner is used only if the previous learners decisions lack confidence.

3 Challenging concepts

None

4 Interesting concepts

Mixture of experts, it was interesting to know that we can actually vary weights depending on the input.

5 Concepts not understood

None

6 Ideas

In ensemble learning we try to select optimal base-learners for classification given any input. What we can do is we can train a $k-NN$ classifier where we group training inputs together based on the base-learner which best classifies them. Then, once this is done, any new instance is first classified by this $k-NN$, and once this is done we know which was the base-learner which best classified this group and use that, or we can also additionally select some nearby learners to this group and use them with appropriate weights.