SL05: Ensemble Learning - Boosting

Introduction:

- Ensemble Learning is in general the process of combining some simple rules into a single more complicated rule that can generalize well.
- It works by dividing the data into smaller subsets, learn over each individual subset to come up with a rule, then combine all these rules in a single more complex rule.
- If we looked at the whole data, it would be hard to come up with simple rules.

Ensemble Learning Algorithm:

- Steps:
  1. Learn over a subset of data: Uniformly randomly pick a subset of the data.
  2. Generate a rule: Apply a learning algorithm to the subset.
  3. Repeat over other subsets.
  4. Combine the rules into a single rule: Take the mean.
- For example, if we have a dataset, and we applied the Ensemble Learning Algorithm described above:
  - Steps:
    1. Split the dataset into 5 subsets.
    2. Apply a 3rd order polynomial to each subset.
    3. Repeat over other subsets.
    4. Take the average of all the output functions (5 curves).
  - It turns out that if we applied a more complex learning rule on the whole dataset (4th order polynomial), it will produce higher accuracy on the training data. On the other hand, the average of the five 3rd order polynomials will produce higher accuracy on the testing data. So basically the Ensemble Learner generalizes better (lower overfitting).
  - This happens because taking subsets of the examples as opposed to looking on the whole dataset averages out the noisy examples. It's similar to what happens with cross validation.

Bagging (Bootstrap Aggregation):

- For example, if we have a dataset, and we applied the Ensemble Learning Algorithm described above.
- Steps:
  1. Split the dataset into 5 subsets.
  2. Apply a 3rd order polynomial to each subset.
  3. Repeat over other subsets.
  4. Take the average of all the output functions (5 curves).

- It turns out that if we applied a more complex learning rule on the whole dataset (4th order polynomial), it will produce higher accuracy on the training data. On the other hand, the average of the five 3rd order polynomials will produce higher accuracy on the testing data. So basically the Ensemble Learner generalizes better (lower overfitting).
- This happens because taking subsets of the examples as opposed to looking on the whole dataset averages out the noisy examples. It's similar to what happens with cross validation.

Boosting:

- Choosing subsets: Rather than selecting random subset like Bagging, we'll learn as we go. We'll choose subsets containing the hardest examples, those ones that don't perform well given the current rule.
- Combining learners: Apply weighted mean instead of normal mean. This will help us avoid ending up with a constant function that doesn't represent the underlying relationship.
- Error:
  - In regression, the error is the squared difference between correct and predicted values.
  - In classification, the error is the ratio of the total number of mismatches over the total number of examples. This implies implicitly that all the examples are equally important.
  - In the case of classification, instead of considering only the number of mismatches, we should also consider the probability that the learner will disagree with the true concept on a particular instance.

$$P_{r_{\mathbb{D}}}[h(x) = c(x)]$$

  - This shifts the perspective from the notion of only being wrong, to the notion of what is the effect of being wrong in a case-by-case investigation.
- Weak learner: A Weak Learner is a learner that no matter what the distribution is, always get an error rate that is better than chance (or better that 50%). It's performance is always slightly better than random guessing.

Boosting in Code:

- Given training data $\{(x_i, y_i)\}$, $y_i \in \{-1, +1\}$
- For $t = 1$ to T:
  - Construct the distribution $\mathbb{D}_t$
  - Find weak classifier $h_t(x)$ with small error $\varepsilon_t = P_{D_t}[h_t(x_i) \neq y_i]$
- Output H_{final}

Distribution:

- The distribution at the beginning will be uniform over all the examples.

$$\mathbb{D}_1(i) = \frac{1}{n}$$

- For each distribution afterwards:

$$\mathbb{D}_{t+1}(i) = \frac{\mathbb{D}_t(i).\, e^{-\alpha_t y_i h_t(x_i)}}{z_t}$$

$$where \qquad \alpha_t = \frac{1}{2}\ln\frac{1-\varepsilon_t}{\varepsilon_t}$$

- We take the old distribution of a particular example, and we make it bigger or smaller based upon how well the current hypothesis does on that example.
- This equation guarantees that If the prediction was correct, the example's probability in the distribution generally decreases (this means the weight will decrease).
- If the learner gets the prediction wrong, the example's probability in the distribution increases (this means the weight will increase), but it ultimately depends on the rest of the distribution and how the learner performed.
- Final hypothesis:

$$H_{final}(x) = sgn\left(\sum_t \propto_t h_t(x)\right)$$

- Alpha is a measure of how well the hypothesis actually performed. The final hypothesis is the weighted sum of the hypotheses multiplied by $\propto_t$.

Why Boosting works?

- The reason why boosting works and produces progressively is that the overall error has to necessarily stay go down with each iteration of boosting, or at least stay the same, due to how the distribution changes in response to getting cases wrong and constantly having to find a weak learner.
- Rule of thumb: Boosting doesn't overfit (Refer to SL06 for explanation).

AdaBoost:

- The AdaBoost algorithm trains multiple weak classifiers on training data, and then combines those weak classifiers into a single "boosted" classifier.
- The combination is done through a weighted sum of the weak classifiers with weights dependent on the weak classifier accuracy.