

Ensemble Learning

Some slides were adapted/taken from various sources, including Prof. Andrew Ng's Coursera Lectures, Stanford University, Prof. Kilian Q. Weinberger's lectures on Machine Learning, Cornell University, Prof. Sudeshna Sarkar's Lecture on Machine Learning, IIT Kharagpur, Prof. Bing Liu's lecture, University of Illinois at Chicago (UIC), CS231n: Convolutional Neural Networks for Visual Recognition lectures, Stanford University, Dr. Luis Serrano, Prof. Alexander Ihler and many more. We thankfully acknowledge them. Students are requested to use this material for their study only and **NOT** to distribute it.

Ensemble Learner

- Multiple Learner
 - Same or different decisions
- Combine decisions
- Generate the group of base learners
 - How to generate
- Base learners are different in ways
 - Different algorithms (for example: SVM, NN, Logistic regression etc.)
 - Same algorithms but with different hyper-parameters
 - Different representations
 - Different modalities
 - Different training sets

Ensemble Learner

- Learning algorithms may have high variance
 - When they are applied on different sub-set of the data, they produce different outputs.
- No learner is absolutely better than other learner
- It is possible to merger a number of weak learners to get a strong learner

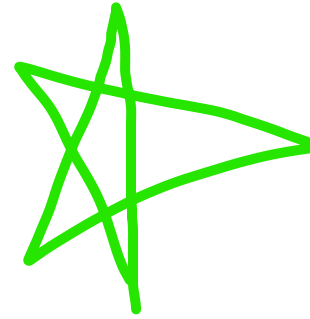
Strong vs. weak learners

- Strong learner: we seek to produce one classifier for which the classification error can be made arbitrarily small
 - So far we were looking for such methods
- Weak learner: a classifier which is just better than random guessing
 - Now this will be our only expectation
- Ensemble learning: instead of creating one strong classifier, we create a huge set of weak classifiers, then we combine their outputs into one final decision
 - According to Concordet's theorem, under proper conditions we can expect that the ensemble model can attain an error rate that is arbitrarily close to zero
 - While creating a lot of weak classifiers is hopefully a much easier task than to create one strong classifier

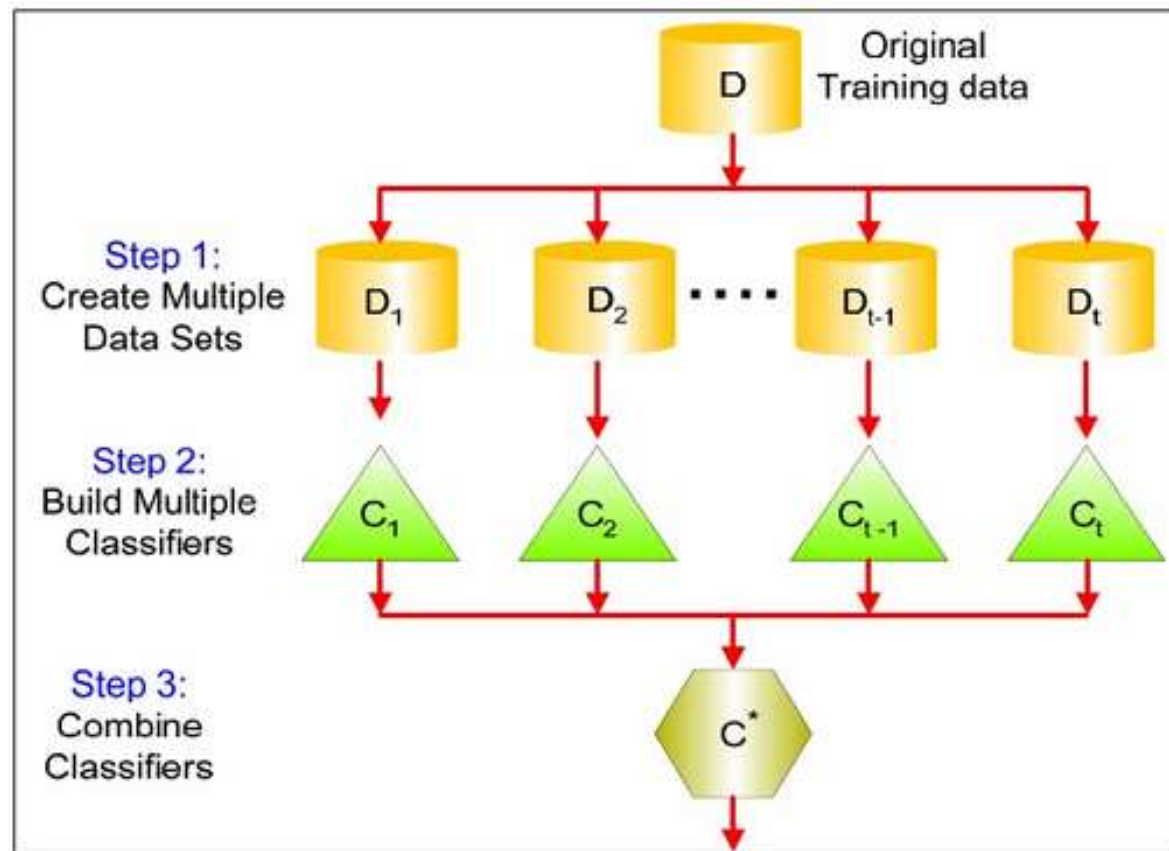
Concordet's theorem:

The error rate of an ensemble of classifiers is always lower than the error rate of the individual classifiers that make up the ensemble, as long as the individual classifiers are not completely independent. In other words, the more the individual classifiers agree with each other, the better the overall accuracy of the ensemble.


This is why it is important to use diverse models with different biases and inputs to create an ensemble, rather than simply using identical models trained on the same data.



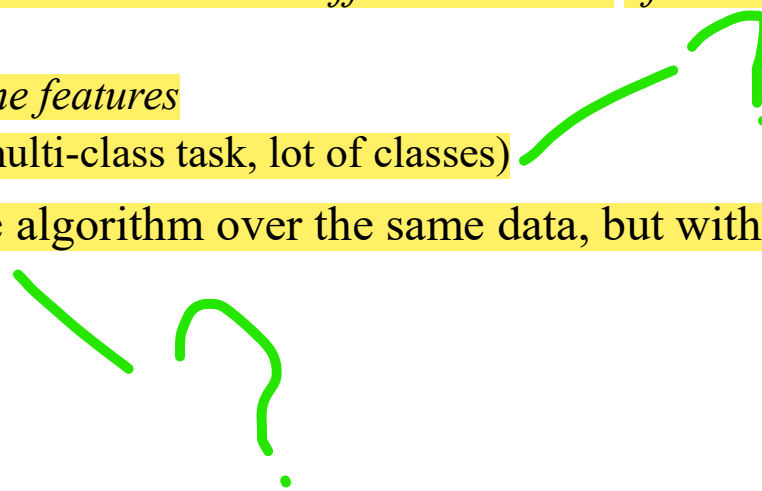
Ensemble Learning



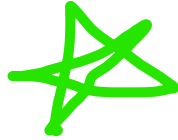
Conditions

- Training the same classifier on the same training data several times would give the same result for most machine learning algorithms
 - Exception: methods where the training involves some randomness
 - Combining these classifiers would make no sense
 - Classifier combination gives the best result if
 - The classifier outputs for the same input are diverse
 - The classifiers operate independently (or at least partially independently)
 - The classifiers have some unique or “local” knowledge
 - E.g. they are trained on different (or at least partially different) training data sets
 - We also must have some formalism to combine (aggregate) the opinions of the various classifiers
- 

How to produce diverse classifiers?

- We can combine *different learning algorithms* (“hybridization”)
 - E.g. we can train a GMM, an SVM, a k-NN,... over the same data, and then combine their output
 - We can combine the same learning algorithm trained several times over the same data
 - This works only if there is some random factor in the training method
 - E.g.: neural networks trained with *different random initialization*
 - We can combine the same learning algorithm trained over *different subsets of the training data*
 - We can also try using *different subsets of the features*
 - Or *different subsets of the target classes* (multi-class task, lot of classes)
 - For certain algorithms we can use the same algorithm over the same data, but with a *different weighting over the data instances*
- 

Bias and Variance



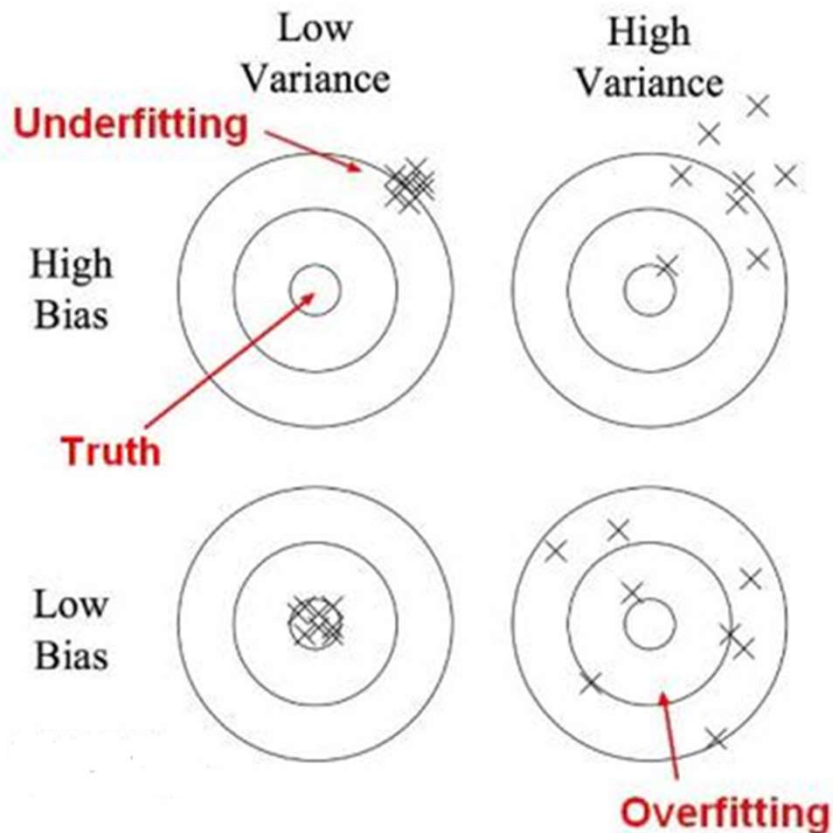
What is bias?

Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with **high bias** pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.

What is variance?

Variance is the variability of model prediction for a given data point or a value which tells us spread of our data. Model with **high variance** pays a lot of attention to training data and does not generalize on the data which it hasn't seen before. As a result, such models perform very well on training data but has high error rates on test data.

Bias and variance using bulls-eye diagram



- Centre of the target is a model that perfectly predicts correct values. As we move away from the bulls-eye our predictions become get worse and worse.

• In supervised learning **underfitting** happens when a model unable to capture the underlying pattern of the data.

- These models usually have high bias and low variance.

- It happens when we have very less amount of data to build an accurate model or when we try to build a linear model with a nonlinear data.

Bias and Variance

- We need low bias and low variance
- Ensemble be multiple trained models
- Assume that these models have low bias and high variance.
- By combining the outputs, we can get low variance while maintaining the low bias
- Even if we start with high bias, by combining we can get low bias.
- Because, individual hypothesis may have high biases but combining them results a new hypothesis function which may be of low bias.
- So, using ensemble, it is possible to achieve low bias and low variance. Thus having less overfitting.

How to combine?

- Ensemble classifier can be obtained by combining different base learners (weak learners i.e. with high error (ϵ) but $\epsilon < 0.5$).
- Now , to combine weak learners, certain conditions should be met
- Independence
 - Let there are n (let say $n=10$) independent learners, each with error 0.3 (accuracy =0.7).
 - Now, if all learners ($n=10$) will give the same output, the confidence score will be $1 - (1 - 0.7)^{10} = 1 - (0.3)^{10}$ i.e. very close to 1 (i.e. quite high)
 - But in practice, all learners may not give identical output. So, we can go for Majority Voting (for two class problem)

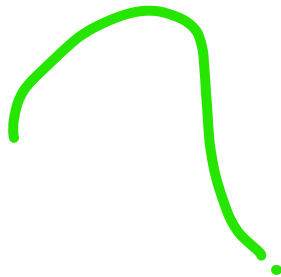
How to combine?

- Let n_1 no. of classifiers output for Class 1 and n_2 no. of classifiers output for Class 2 and ($n_1 > n_2$)
- Probability of class 1 is calculated as follows

$$P(\text{class 1}) = 1 - \text{Binomial}(n, n_2, 0.7)$$

$$\text{where } P(r) = n_{C_r} p^r (1 - p)^{n-r}$$

$$= n_{C_{n_2}} (0.7)^{n_2} (1 - 0.7)^{n_1} = n_{C_{n_2}} (0.7)^{n_2} (0.3)^{n_1}$$



Majority Voting

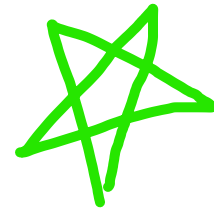
- Unweighted voting (equal weight to every classifiers)
- Weighted voting (weight \propto accuracy, weight $\propto \frac{1}{\text{variance}}$)

$$y = \sum_{j=1}^N w_j d_j$$

where $w_j \geq 0$ and $\sum_j w_j = 1$

- Bayesian way of ensemble:

$$P(C_i|x) = \sum_{\text{all Models } M_j} P(C_i|x, M_j)P(M_j)$$



Bayesian Model Averaging (BMA)

- Bayesian way of ensemble:

$$P(C_i|x) = \sum_{\text{all Models } M_j} P(C_i|x, M_j)P(M_j)$$

- There are not practically feasible
 - Large number of hypothesis in the hypothesis space
 - Prior probability ($P(M_j)$) of model is not always available

$$y = \underset{c_j \in \mathcal{C}}{\operatorname{argmax}} \sum_{h_i \in H} P(c_j|h_i)P(T|h_i)P(h_i)$$

where T is the training samples

- From the Bayesian perspective, it gives a optimal hypothesis

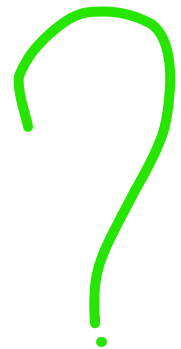
Independence of the learners

- If the outputs of the different learners are independent, the variance get reduced.

$$\text{var}(y) = \text{var}\left(\sum_j \frac{1}{N} d_j\right) = \frac{1}{N^2} N \text{var}(d_j) = \frac{1}{N} \text{var}(d_j)$$

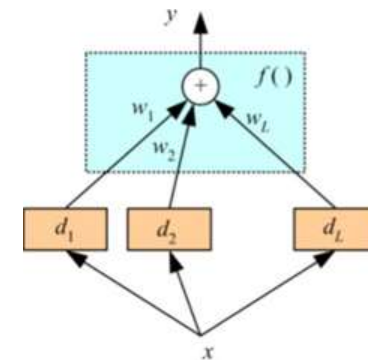
- If the learners are not completely independent but not completely correlated

$$\text{var}(y) = \frac{1}{N} [\text{var}(d_j) + 2 \sum_i \sum_{j>i} \text{cov}(d_i, d_j)]$$



Summary

- There are several methods to combine (aggregate) the outputs of the various classifiers
- When the output is a class label:
 - Majority voting
 - Weighted majority voting (e.g we can weight each classifier by its reliability (which also has to be estimated somehow, of course...))
- When the output is numeric (e.g. a probability estimate for each class c_i):
 - We can combine the d_i scores by taking their (weighted) mean, product, minimum, maximum, ...
- Stacking
 - Instead of using the above simple aggregation rules, we can train yet another classifier on the output values of the base classifiers



Rule	Fusion function $f(\cdot)$
Sum	$y_i = \frac{1}{L} \sum_{j=1}^L d_{ji}$
Weighted sum	$y_i = \sum_j w_j d_{ji}, w_j \geq 0, \sum_j w_j = 1$
Median	$y_i = \text{median}_j d_{ji}$
Minimum	$y_i = \min_j d_{ji}$
Maximum	$y_i = \max_j d_{ji}$
Product	$y_i = \prod_j d_{ji}$

Bagging and Boosting

Bagging

- Bagging = **B**ootstrap + **a**ggregating
- It uses bootstrap resampling to generate L different training sets from the original training set
- On the L training sets it trains L base learners
- During testing it aggregates the L learners by taking their average (using uniform weights for each classifiers), or by majority voting
- The diversity or complementarity of the base learners is not controlled in any way, it is left to chance and to the instability of the base learning method
- The ensemble model is almost always better than the unique base learners if the base learners are unstable (which means that a small change in the training dataset may cause a large change in the result of the training)

Bootstrap resampling

- Suppose we have a training set with n samples
- We would like to create L different training sets from this
- Bootstrap resampling takes random samples from the original set with replacement
- Randomness is required to obtain different sets for L rounds of resampling
- Allowing replacement is required to be able to create sets of size n from the original data set of size n
- As the L training sets are different, the result of the training over these set will also be more or less different, independent of what kind of training algorithm we use
 - Works better with unstable learners (e.g. neural nets, decision trees)
 - Not really effective with stable learners (e.g. k-NN, SVM)

Boosting

- Bagging created a diversity of base learners by creating different variants of the training dataset randomly
 - However, we do not have direct control over the usefulness of the newly added classifiers
- We would expect a better performance if the learners also complemented each other
 - They would have “expertise” on different subsets of the data
 - So they would work better on different subsets
- The basic idea of boosting is to generate a series of base learners which complement each other
 - For this, we will force each learner to focus on the mistakes of the previous learner

Boosting

- We represent the importance of each sample by assigning weights to the samples
 - Correct classification \rightarrow smaller weights
 - Misclassified samples \rightarrow larger weights
- The weights can influence the algorithm in two ways
 - Boosting by sampling: the weights influence the resampling process
 - This is a more general solution
 - Boosting by weighting: the weights influence the learner
 - Works only with certain learners
- Boosting also makes the aggregation process more clever: We will aggregate the base learners using weighted voting
 - Better weak classifier gets a larger weight
 - We iteratively add new base learners, and iteratively increase the accuracy of the combined model

AdaBoost (Adaptive Boosting)

- Boosting can turn a weak algorithm into a strong learner.
- Input: $S = \{(x_1, y_1), \dots, (x_m, y_m)\}$
- $D_t(i)$: weight of i th training example
- Weak learner A
- For $t = 1, 2, \dots, T$
 - Construct D_t on $\{x_1, x_2, \dots\}$
 - Run A on D_t producing $h_t: X \rightarrow \{-1, 1\}$
 - ϵ_t = error of h_t over D_t

AdaBoost (Adaptive Boosting)

Given: $(x_1, y_1), \dots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$

Initialize $D_1(i) = 1/m$.

For $t = 1, \dots, T$:

- Train weak learner using distribution D_t .
- Get weak classifier $h_t: X \rightarrow \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Where Z_t is a normalization factor

$$Z_t = \sum_{i=1}^m D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$

Output the final classifier:

$$H(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right).$$

AdaBoost (Adaptive Boosting)

Choose α_t to minimize training error

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

where

$$\epsilon_t = \sum_{i=1}^m D_t(i) \delta(h_t(x_i) \neq y_i)$$

to continue...