Ensemble Models

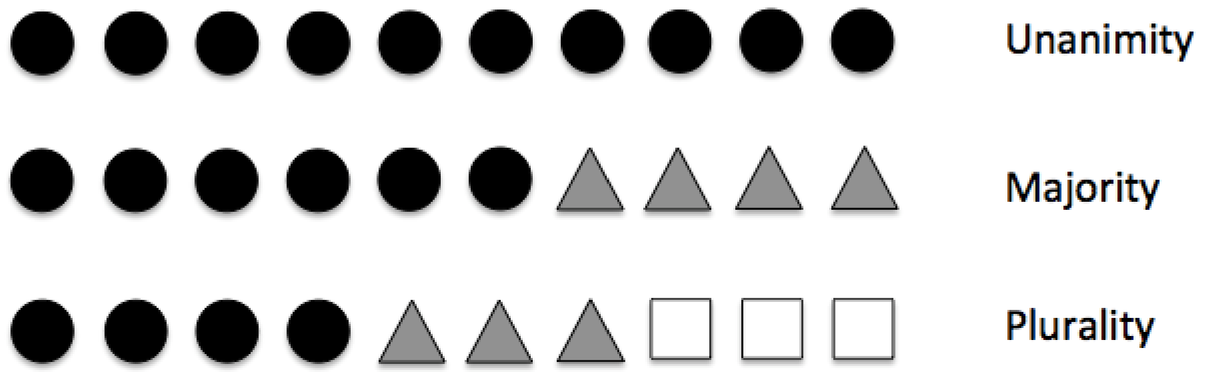
PhD Abay Nussipbekov

Learning with ensembles

- Our goal is to combine multiple classifiers
- Mixture of experts, e.g. 10 experts
- Predictions more accurate and robust
- Provide an intuition why this might work
- Simplest approach: majority voting

Majority voting

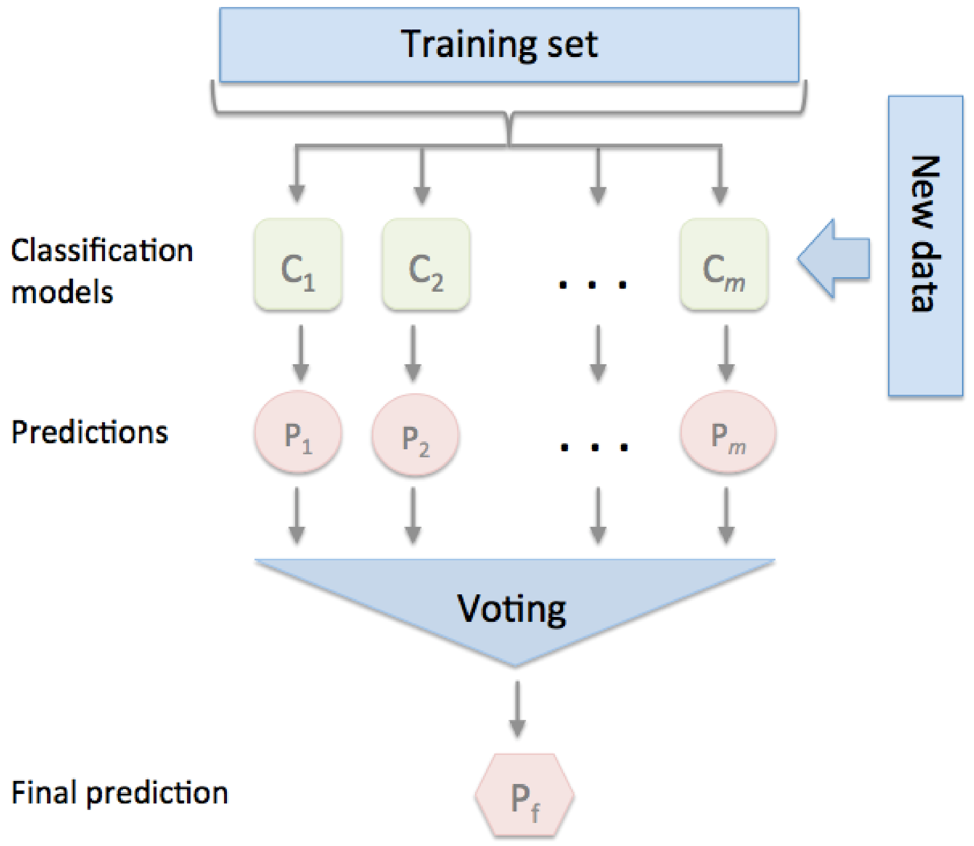
- Majority voting refers to binary setting
- Can easily generalize to multi-class: plurality voting
- Select class label that receives the most votes (mode)



Combining predictions: options

- Train m classiers $C_1, \ldots, C_m$
- Build ensemble using different classification algorithms (e.g. SVM, logistic regression, etc.)
- Use the same algorithm but with different subsets of the training set (e.g. random forest)

General approach



Combining predictions via majority voting

- We have predictions of individual classifiers C_j and need to select the final class label $\hat{y}$

$$\hat{y} = model\{C_1(x), C_2(x), \ldots, C_m\}$$

- For example, in a binary classification task where class1 = -1 and

class2 = +1, we can write the majority vote prediction as follows:

$$C(x) = sign\left[\sum_{j}^{m} C_j(x)\right] = \begin{cases} 1\ if\ \sum_j C_j(x) \geq 0 \\ -1 \qquad otherwise \end{cases}$$

Intuition why ensembles can work better

- Assume that all n base classifiers have the same error rate ϵ. We can express the probability of an error of an ensemble can be expressed as a probability mass function of a binomial distribution:

$$P(y \geq k) = \sum_{k}^{n} \binom{n}{k} \epsilon^{k} (1 - \epsilon)^{n-k} = \epsilon_{ensemble}$$

- Here, $\binom{n}{k}$ is the probability coefficient *n choose k*. In other words, we compute the probability that the prediction of the ensemble is wrong.

Example

- Imagine we have 11 base classifier (n=11) with an error rate of 0.25 ($\epsilon = 0.25$):

$$P(y \geq k) = \sum_{k=6}^{11} \binom{11}{k} 0.25^k (1 - 0.25)^{11-k} = 0.034$$

- So the error rate of the ensemble of n = 11 classifiers is much lower than the error rate of the individual

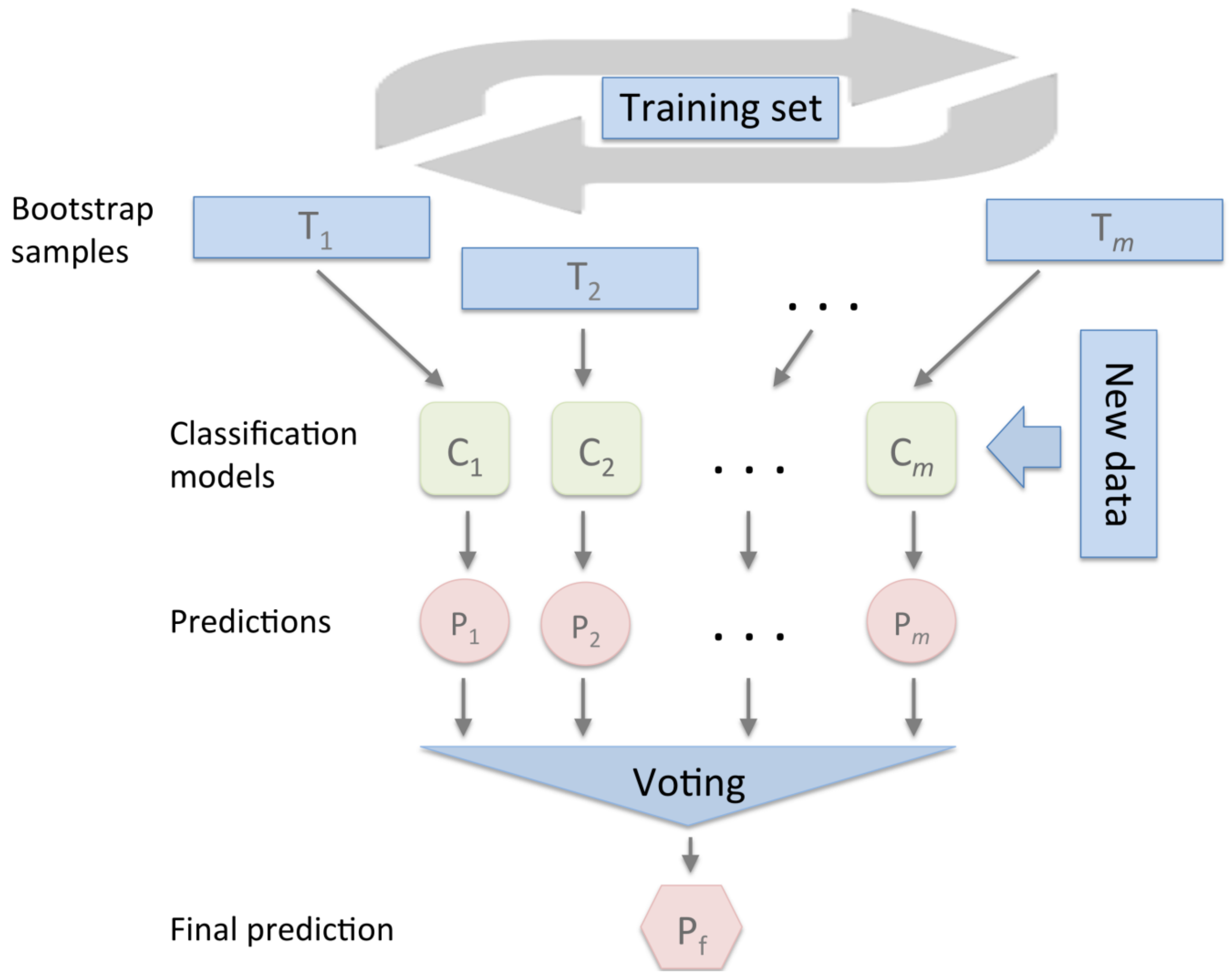
Voting classifier in scikit-learn

- Simply instantiate several classifiers
- Make a list
- Pass to sklearn.ensemble.VotingClassifier(…)
- ROC Curves and Area Under the Curve
- Ipython code

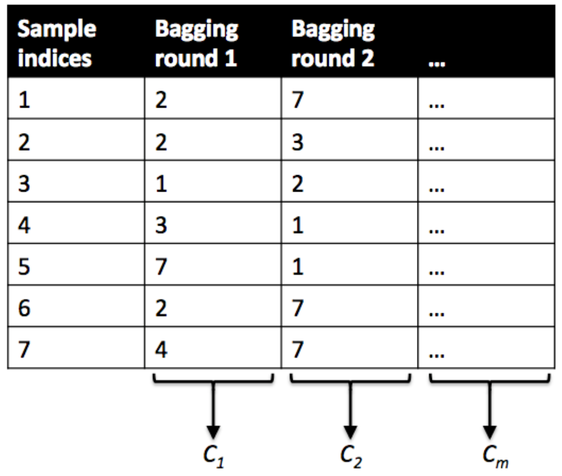
Bootstrap aggregation (bagging)

- We used the entire training set for the majority vote classifier
- Here we draw bootstrap samples
- In statistics, bootstrapping is any test or metric that relies on random sampling with replacement.
- Hypothesis testing: bootstrapping often used as an alternative to statistical inference based on the assumption of a parametric model when that assumption is in doubt
- The basic idea of bootstrapping is that inference about a population from sample data, can be modelled by resampling with replacement the sample data and performing inference about a sample from resampled data.

Bagging


Training set
Bootstrap samples
T_1
T_2
. . .
T_m
New data
Classification models
C_1
C_2
. . .
C_m
Predictions
P_1
P_2
. . .
P_m
Voting
Final prediction
P_f

Bootstrapping example



Sample indices	Bagging round 1	Bagging round 2	...
1	2	7	...
2	2	3	...
3	1	2	...
4	3	1	...
5	7	1	...
6	2	7	...
7	4	7	...

- Seven training examples
- Sample randomly with replacement
- Use each bootstrap sample to train a classier Cj
- Cj is typically a decision tree
- Random Forests: also use random feature subsets

Random forest intuitive explanation

- Choose a movie to watch. Ask Baurzhan.
- You name him movies you liked or not (training).
- 20 question game to figure out if you like a movie or not: “Is X a romantic movie”? “Does Jonny Depp star in X?” and etc. Baurzhan is a decision tree.
- To generalize you ask other friends (Dina, Temirlan, Azamat, etc.). Ensemble classifier - forest.
- Give different data to friends – may be you don’t actually like Titanic (were happy that day), may be you really really love Forest Gump so you want other friends give more weight on it (watched two times).
- You don’t change decision, you just change say you love/hate some movie more/less – bootstrapping data.
- May be you loved Titanic and Inception not because of Leonardo DiCaprio, so you don’t allow for some friends to use him as a question – randomly picking features.
- And so your friends now form a random forest.

Bagging in scikit-learn

- Instantiate a decision tree classier
- Make a bagging classier with decision trees
- Check that the accuracy is higher for the bagging classier
- Ipython example

Boosting

- Basic idea: start with weak learners that have only a slight performance advantage over random guessing (e.g. a decision tree stump) and try to boost their performance by focusing on training samples that are hard to classify
- Very simple base classifiers learn from missclassified training
- examples
- AdaBoost (short for Adaptive Boosting) is the most common implementation of boosting

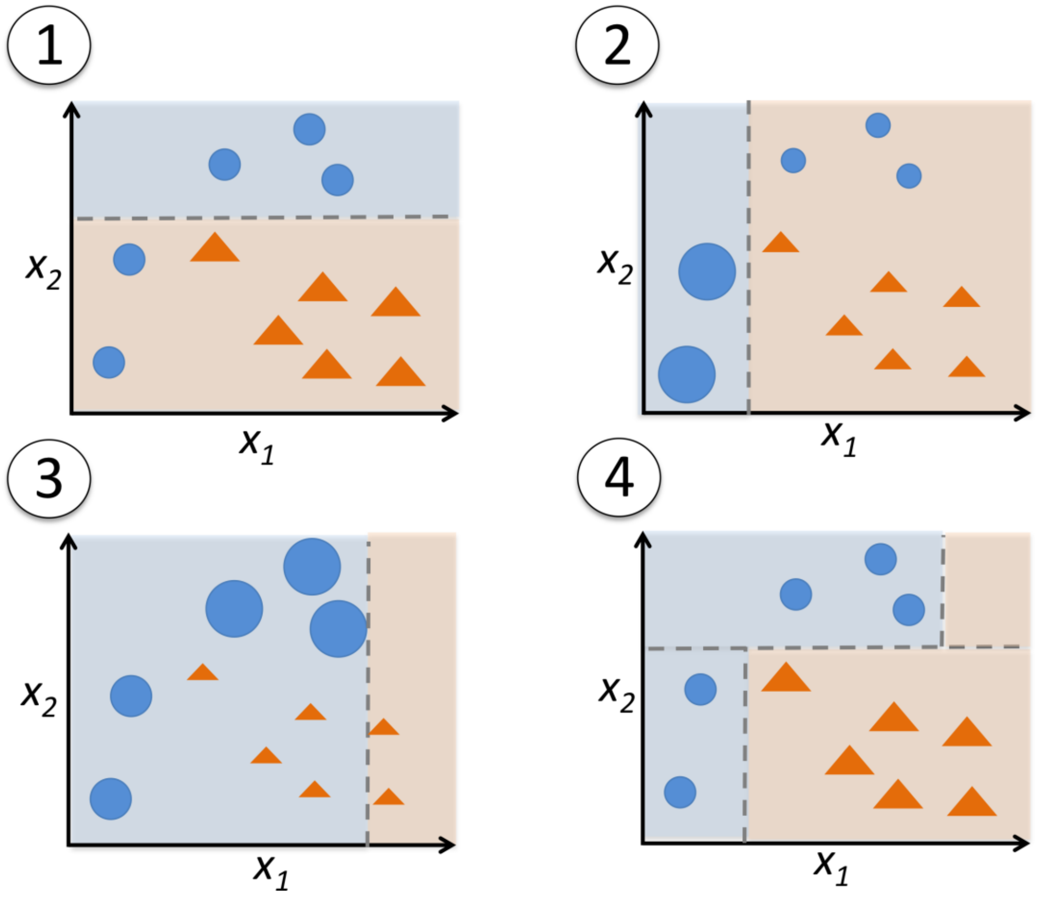
Original boosting algorithm

1. Draw a random subset of training samples d1 without replacement from the training set D to train a weak learner C1
2. Draw second random training subset d2 without replacement from the training set and add 50 percent of the samples that were previously misclassified to train a weak learner C2
3. Find the training samples d3 in the training set D on which C1 and C2 disagree to train a third weak learner C3
4. Combine the weak learners C1, C2, and C3 via majority voting

AdaBoost

- In contrast, AdaBoost uses the complete training set to train the weak learners
- Training samples are reweighted in each iteration to build a strong classifier
- End goal is to build a strong classier that learns from the mistakes of the previous weak learners in the ensemble

AdaBoost



AdaBoost algorithm

1. Set weight vector ***w*** to uniform weights where $\sum_i w_i = 1$
2. For *j* in *m* boosting rounds, do the following:
    1. Train a weighted weak learner: $C_j = train(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{w})$
    2. Predict class labels: $\hat{y} = predict(C_j, \boldsymbol{X})$
    3. Compute weighted error rate: ε = ***w***·(***y***ˆ ≠ ***y***) .
    4. Compute the coefficient: $\alpha_j = 0.5log\frac{1-\epsilon}{\epsilon}$
    5. Update the weights **w = w**$\times$ $\exp(-\alpha_j \times \hat{\boldsymbol{y}} \times \boldsymbol{y})$
    6. Normalize weights to sum to 1: **w** = **w** / $\sum_i w_i$
3. Compute final prediction:

$$\hat{y} = (\sum_{j=1}^{m} \left(\alpha_j \times predict\left(C_j, \boldsymbol{X}\right)\right) > 0)$$