Chapter 7

Ensemble Classifiers

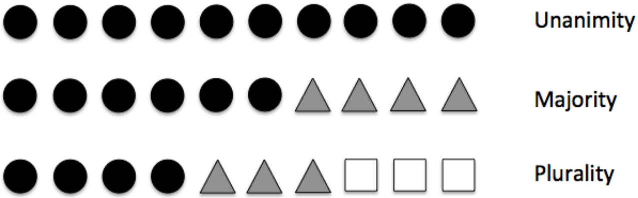
August 11, 2017

Learning with ensembles

- Our goal is to combined 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

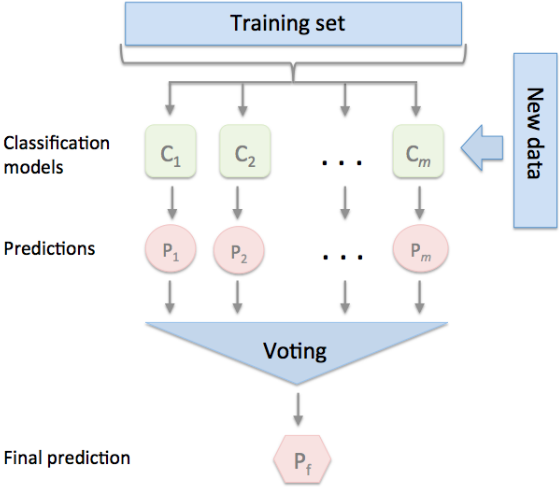
- Majority voting refers to binary setting
- Can easily generalize to multi-class: plurality voting
- Select class label that receives the most votes (mode)



Combining predictions: options

- Train m classifiers $C_1, \ldots, C_m$
- Build ensemble using different classification algorithms (e.g. SVM, logistic regression, etc.)
- Use the same algorithm but fit 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 $\hat{y}$

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

For example, in a binary classification task where $class_1 = -1$ and $class_2 = +1$, we can write the majority vote prediction as follows:

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

Intuition why ensembles can work better

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

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

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

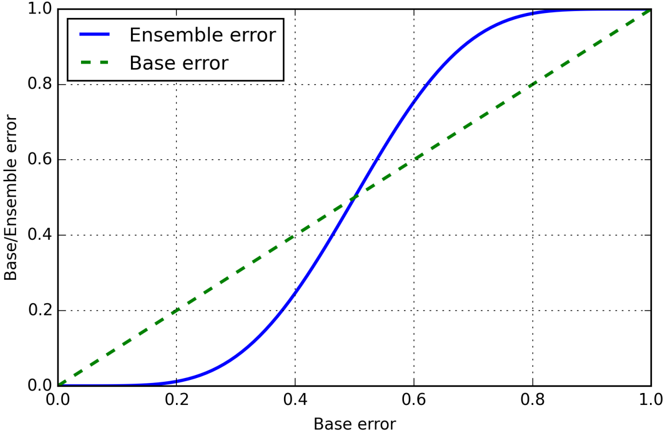
Example

Imagine we have 11 base classifiers ($n = 11$) 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 classifiers.

Same reasoning applied to a wider range of error rates



Voting classifier in scikit-learn

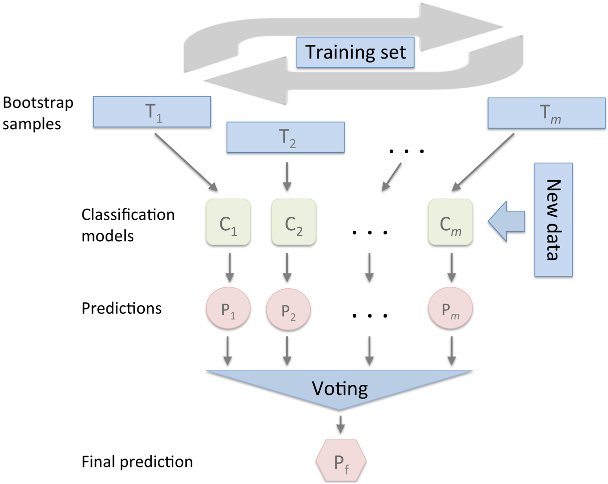
- Simply instantiate several classifiers
- Make a list
- Pass to sklearn.ensemble.VotingClassifier(...)
- API link

```
clf1 = LogisticRegression(random_state=1)
clf2 = RandomForestClassifier(random_state=1)
clf3 = GaussianNB()
estimators=[('lr', clf1), ('rf', clf2), ('gnb', clf3)]
ens_clf = VotingClassifier(estimators)
ens_clf = eclf1.fit(X, y)
```

Boostrap 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
Classification models
C_1
C_2
C_m
New data
Predictions
P_1
P_2
P_m
Voting
Final prediction
P_f

Boostrapping 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	...

C_1 C_2 C_m

- Seven training examples
- Sample randomly with replacement
- Use each boostrap sample to train a classifier C_j
- C_j is typically a decision tree
- **Random Forests**: also use random feature subsets

Bagging in scikit-learn

- Instantiate a decision tree classifier
- Make a bagging classifier with decision trees
- Check that the accuracy is higher for the bagging classifier
- PML github

- 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 misclassified training examples
- The original boosting algorithm was formulated by Robert Schapire in 1990
- It was later refined into **AdaBoost**
- **AdaBoost** (short for Adaptive Boosting) is the most common implementation of boosting

Original boosting algorithm

1. Draw a random subset of training samples d_1 without replacement from the training set D to train a weak learner C_1
2. Draw second random training subset d_2 without replacement from the training set and add 50 percent of the samples that were previously misclassified to train a weak learner C_2
3. Find the training samples d_3 in the training set D on which C_1 and C_2 disagree to train a third weak learner C_3
4. Combine the weak learners C_1, C_2, and C_3 via majority voting

- 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 classifier that learns from the mistakes of the previous weak learners in the ensemble

AdaBoost algorithm

1. Set weight vector $\mathbf{w}$ 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: $C_j = train(\mathbf{X}, \mathbf{y}, \mathbf{w})$.
    2. Predict class labels: $\hat{y} = predict(C_j, \mathbf{X})$.
    3. Compute the weighted error rate: $\epsilon = \mathbf{w} \cdot (\hat{\mathbf{y}} \neq \mathbf{y})$.
    4. Compute the coefficient α_j: $\alpha_j = 0.5 \log \frac{1-\epsilon}{\epsilon}$.
    5. Update the weights: $\mathbf{w} := \mathbf{w} \times \exp\big(-\alpha_j \times \hat{\mathbf{y}} \times \mathbf{y}\big)$.
    6. Normalize weights to sum to 1: $\mathbf{w} := \mathbf{w} / \sum_i w_i$.
3. Compute the final prediction:
$\hat{\mathbf{y}} = \big(\sum_{j=1}^{m} \big(\alpha_j \times predict(C_j, \mathbf{X})\big) > 0\big)$.

Notes: For clarity, we will denote element-wise multiplication by the cross symbol $(\times)$ and the dot product between two vectors by a dot symbol $(\cdot)$, respectively. Note that the expression $(\hat{\mathbf{y}} == \mathbf{y})$ in step 5 refers to a vector of 1s and 0s, where a 1 is assigned if the prediction is incorrect and 0 is assigned otherwise.