Chapter 12

Ensemble Methods

Ensemble methods. Bagging and bragging. Boosting and stumping.

12.1 Bagging

Ensemble methods combine different hypotheses, whether from regression models or classifiers, in order to improve prediction. One way of doing this is to train different instances of some model with different training sets and then aggregate the response, either with an average, for regression problems, or with majority voting for classification. We can obtain replicas of the training set with bootstrapping, as we saw on Chapter 11, and use those to train different hypotheses. In the code below, we first create a vector pxof x values to plot our polynomial curves and a matrix for all the predictions. Then we fit the polynomial model to each replica and compute the predicted values. Finally, we compute the mean of each prediction. This method is called *bootstrap aggregating* or *bagging*, for short.

```
train_sets, _ = bootstrap(replicas,data)
px = np.linspace(ax_lims[0],ax_lims[1],points)
preds = np.zeros((replicas,points))
for ix in range(replicas):
    coefs = np.polyfit(train_sets[ix,:,0], train_sets[ix,:,1],degree)
    preds[ix,:] = np.polyval(coefs,px)
mean = np.mean(preds,axis=0).ravel()
```

Alternatively, we can also use the median instead of the mean for the ensemble. This variant of *bagging* is called *bragging*. Figure 12.1 compares these two variants on a polynomial regression problem. The ensemble of curves was computed using the replicas obtained by bootstrapping.

For classification with *bagging*, instead of averaging the predicted value, the class is predicted by majority voting among the classifiers in the ensemble. Apart from this, the procedure is identical: train the model on a number of replicas of the training set, obtained by bootstrapping. Then apply each resulting classifier and classify according to the class that was given the most times. The code below shows how to create an ensemble of SVM classifiers and then use it to classify new points.

```
train_sets,_ = bootstrap(replicas,data)
gamma = 2
C=10000
svs = []
```

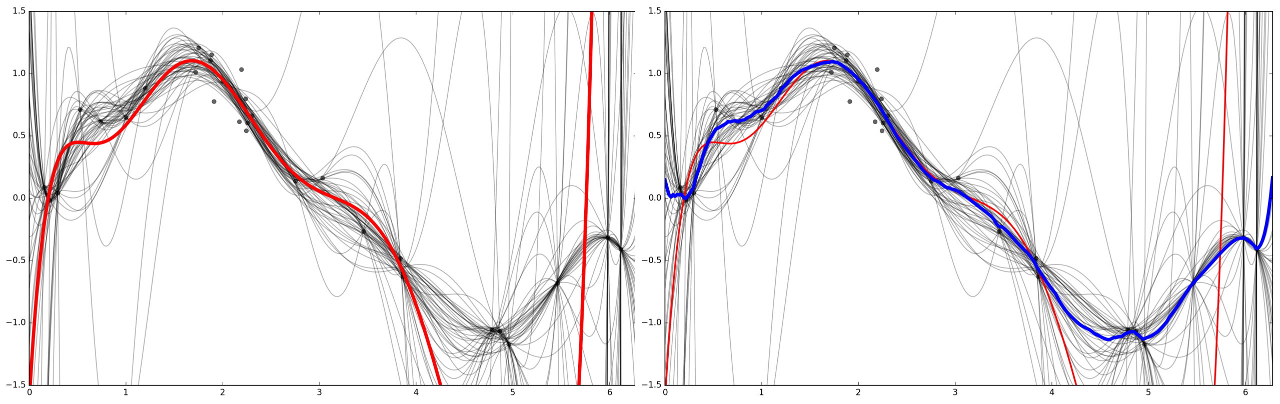

Figure 12.1: The two panels show the instances obtained by training the polynomial model with different replicas of the data set. The left panel shows the ensemble predictions using the mean (*bagging*). The right panel also shows the result using the median (*bragging*).

```
 5 pX,pY = np.meshgrid(pxs,pys)
 6 pZ = np.zeros((len(pxs),len(pys)))
 7 for ix in range(replicas):
 8     sv = svm.SVC(kernel='rbf', gamma=gamma,C=C)
 9     sv.fit(train_sets[ix,:,:-1],train_sets[ix,:,-1])
10     svs.append(sv)
11     preds = sv.predict(np.c_[pX.ravel(),pY.ravel()]).reshape(pZ.shape)
12     pZ = pZ + preds
13 pZ = np.round(pZ/float(replicas))
```

The meshgrid call on line 5 generates the pX and pY matrices for plotting the contour, along with the pZ matrix with the prediction values. Then, for each replica, we train a new SVM classifier and add its predictions to pZ. The majority class, 0 or 1, is computed by rounding the average classification. Figure 12.2 shows the 50 SVM classifiers computed from the replicas of the data-set and the result of the ensemble classifier, using the majority vote from the 50 SVM to classify each point.

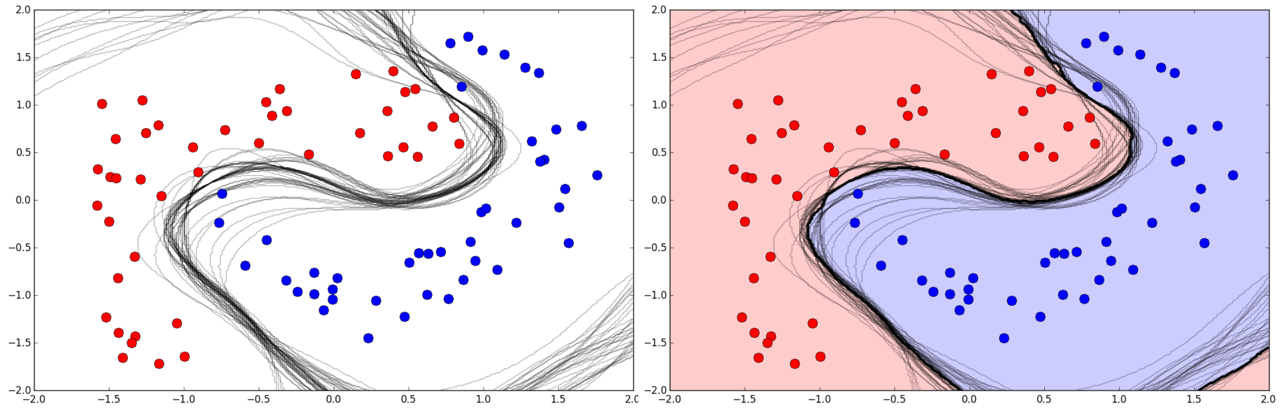

Figure 12.2: The two panels show the 50 SVM classifiers and the resulting ensemble classifier.

Bootstrap aggregating reduces variance without increasing bias and so it is useful if the base model has a high variance and low bias. In classification, the probability of the ensemble classifying an example correctly increases rapidly with the number of classifiers aggregated. For an ensemble of T classifiers, each with a probability p of correctly classifying an example, the probability of a correct classification with majority voting is:

$$\sum_{k=T/2+1}^{T} \binom{T}{k} p^k (1-p)^{T-k}$$

Figure 12.3 shows this increase for different values of p. Even modest classifiers can become quite accurate if aggregated in large ensembles.

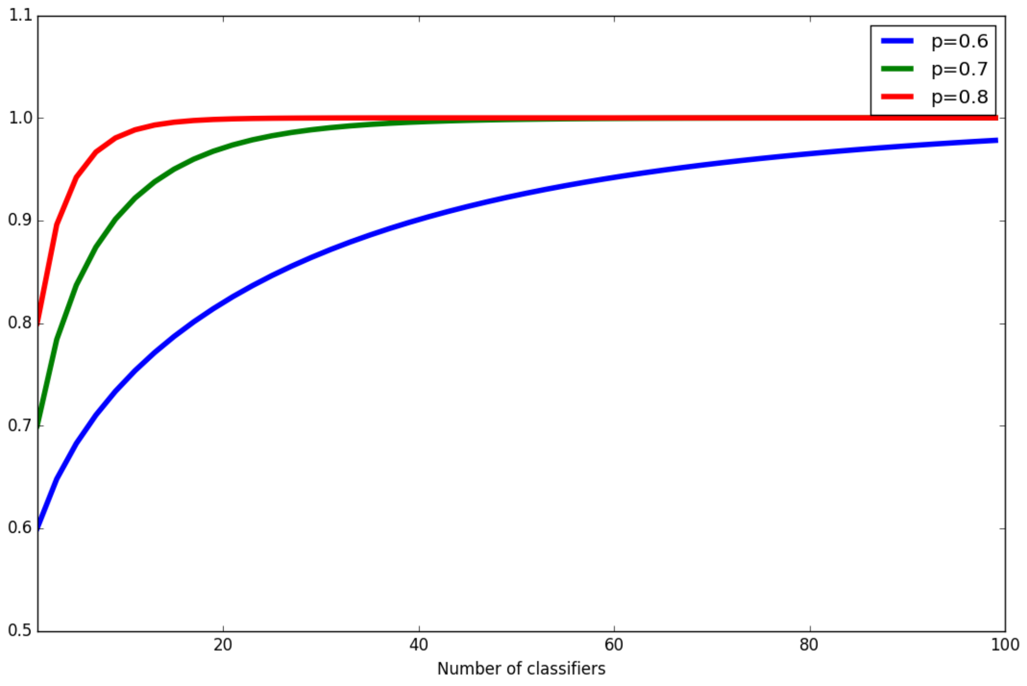

Figure 12.3: Probability of correct classification for different values of p as a function of the number of classifiers in the *bagging* ensemble.

However, this increase in probability presumes that the classifiers are statistically independent. The more correlation there is between classifiers the smaller the improvement gained by the ensemble. This is why it is important to use *unstable classifiers* with a high variance, otherwise they will all be similar and there will be no advantage to aggregating them. *Unstable classifiers* are classifiers that are sensitive to changes in the training set. *Stable classifiers*, such as SVM with a small regularization constant (C) or k-NN, are not good choices for *bagging*.

12.2 Boosting

For *boosting* we'll consider the *adaptive boosting* algorithm, or *AdaBoost*. This method finds a linear combination of weak classifiers, where each individual classifier is assigned a weight, so that the weighted average of the classifier responses minimizes the classification error. This is done by training each classifier with the same data set but giving different weights to different data points, weighing more strongly those that were previously misclassified.

So, first, we initialize the weights of all N examples to $w_n = 1/N$. Then we train one classifier so that we minimize the weighted error of the training set:

$$J_m = \sum_{n=1}^{N} w_m^n I(y_m(x^n) \neq t^n)$$

Then we compute the weighted error of the classifier and the weight of the classifier in the ensemble classifier.

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_m^n I(y_m(x^n) \neq t^n)}{\sum_{n=1}^{N} w_m^n} \qquad \alpha_m = \ln \frac{1-\epsilon_m}{\epsilon_m}$$

Then we use α_m to update the weights for the data points [1]:

$$w_{m+1}^n = w_m^n \exp\left(\alpha_m I(y_m(x^n) \neq t^n)\right)$$

The indicator function I returns 1 if the values are different, 0 if they are equal, so this results in increasing the weights of misclassified points. Then a new classifier is fitted to the training set with the updated weights, and the process is repeated until the weighted training error is zero or greater than 0.5. To use the final ensemble classifier obtained with *AdaBoost* we compute the weighted sum of the responses of the individual classifiers: $f(x) = sign \sum_{m=1}^{M} \alpha_m y_m(x)$

For an example, we'll use a *decision stump* as the weak classifier. This is a decision tree with only one level, and consists of finding the threshold value of a single feature in the data that best splits the classes we wish to separate. For a data set with two dimensions, this means we are creating horizontal and vertical lines to split the data. *Adaptive boosting* with *decision stumps* is called *stumping*.

At each iteration of the *AdaBoost* algorithm, we fit a decision tree classifier with depth of 1 to the weighted data set (line 7), compute the weighted error (lines 8 and 9) and update the weights, storing the value of α_m, computed in line 10. Line 12 normalizes the weights so that they add up to 1.

```
 1 from sklearn.tree import DecisionTreeClassifier
 2 hyps = []
 3 hyp_ws = []
 4 point_ws = np.ones(data.shape[0])/float(data.shape[0])
 5 max_hyp = 50
 6 for ix in range(max_hyp):
 7     stump = DecisionTreeClassifier(max_depth=1)
 8     stump.fit(data[:,:-1], data[:,-1], sample_weight = point_ws)
 9     pred = stump.predict(data[:,:-1])
10     errs = (pred != data[:,-1]).astype(int)
11     err = np.sum(errs*point_ws)
12     alpha = np.log((1-err)/err)
13     point_ws = point_ws*np.exp(alpha*errs)
14     point_ws = point_ws/np.sum(point_ws)
```

Figure 12.4 shows the first three iterations and the final classifier, after 10 iterations. The points are shown in different sizes depending on their current weights.

To classify new points and compute the error, we iterate through the stored classifiers, compute the prediction of each (line 3) and add it, weighted by the respective weight of that classifier (line 4).

```
1 net_pred = np.zeros(data.shape[0])
2 for ix in range(len(hyps)):
```

[1] In the original algorithm, by Freund and Schapire, it was $\alpha_m = \frac{1}{2} \ln \frac{1-\epsilon_m}{\epsilon_m}$. However, this $\frac{1}{2}\alpha$ just affects the weights of the classifier by a constant and the weights of the points by a value that is independent of the point, so can be omitted

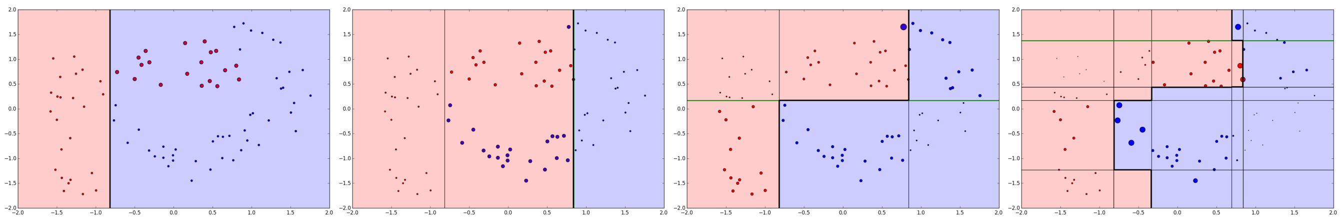
Figure 12.4: Adaptive boosting using decision stumps.

```
    pred_n = hyps[ix].predict(data[:,:-1])
    preds = preds+pred_n*hyp_ws[ix]
net_pred[preds<0] = -1
net_pred[preds>=0] = 1
errors = np.sum((net_pred !=data[:,-1]).astype(int))
```

AdaBoost can be seen as a sequential minimization of an exponential function of the weighted error:

$$E = \sum_{n=1}^{N} \exp\left(-t_n f_m(x_n)\right) \qquad f(x) = \sum_{m=1}^{M} \alpha_m y_m(x)$$

It is sequential because, at each step m, we assume the classifiers and weights for all steps $1, \ldots, m-1$ are fixed, and so the error function at each step is simply:

$$E = \sum_{n=1}^{N} w_m^n \exp\left(-t_n \alpha_m y_m(x_n)\right)$$

which we can decompose by separating the terms corresponding to points that are correctly classified, with $t_n = y_m(x_n)$, and those that are not correcly classified, with $t_n \neq y_m(x_n)$. Letting set $\mathcal{T}$ be the set of points correctly classified by classifier m and set $\mathcal{M}$ the set of points misclassified by classifier m, the error function is:

$$E = e^{-\alpha_m} \sum_{n \in \mathcal{T}} w_m^n + e^{\alpha_m} \sum_{n \in \mathcal{M}} w_m^n = e^{-\alpha_m} \sum_{n=1}^{N} w_m^n + \left(e^{\alpha_m} - e^{-\alpha_m}\right) \sum_{n=1}^{N} w_m^n I(y_m(x^n) \neq t^n)$$

where function I is the indicator function that returns 1 when the point is misclassified or 0 if it is classified correctly. Minimizing as a function of y_m and α_m, we obtain the following solutions:

$$J_m = \sum_{n=1}^{N} w_m^n I(y_m(x^n) \neq t^n) \qquad \alpha_{m+1} = \ln \frac{1-\epsilon_m}{\epsilon_m}$$

which correspond to the weighted error function for *AdaBoost* and the expression for computing α_m.

12.3 Further Reading

1. Alpaydin [2], Sections 17.6 and 17.7
2. Marsland [17], Chapter 7

3. Bishop [4], 14.2, 14.3

Bibliography

[1] Uri Alon, Naama Barkai, Daniel A Notterman, Kurt Gish, Suzanne Ybarra, Daniel Mack, and Arnold J Levine. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proceedings of the National Academy of Sciences*, 96(12):6745–6750, 1999.

[2] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.

[3] David F Andrews. Plots of high-dimensional data. *Biometrics*, pages 125–136, 1972.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, New York, 1st ed. edition, oct 2006.

[5] Deng Cai, Xiaofei He, Zhiwei Li, Wei-Ying Ma, and Ji-Rong Wen. Hierarchical clustering of www image search results using visual. Association for Computing Machinery, Inc., October 2004.

[6] Guanghua Chi, Yu Liu, and Haishandbscan Wu. Ghost cities analysis based on positioning data in china. *arXiv preprint arXiv:1510.08505*, 2015.

[7] Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems*, pages 396–404. Morgan Kaufmann, 1990.

[8] Pedro Domingos. A unified bias-variance decomposition. In *Proceedings of 17th International Conference on Machine Learning. Stanford CA Morgan Kaufmann*, pages 231–238, 2000.

[9] Hakan Erdogan, Ruhi Sarikaya, Stanley F Chen, Yuqing Gao, and Michael Picheny. Using semantic analysis to improve speech recognition performance. *Computer Speech & Language*, 19(3):321–343, 2005.

[10] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[11] Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

[12] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[13] Patrick Hoffman, Georges Grinstein, Kenneth Marx, Ivo Grosse, and Eugene Stanley. Dna visual and analytic data mining. In *Visualization'97., Proceedings*, pages 437–441. IEEE, 1997.

[14] Chang-Hwan Lee, Fernando Gutierrez, and Dejing Dou. Calculating feature weights in naive bayes with kullback-leibler measure. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 1146–1151. IEEE, 2011.

[15] Stuart Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.

[16] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[17] Stephen Marsland. *Machine Learning: An Algorithmic Perspective*. Chapman & Hall/CRC, 1st edition, 2009.

[18] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[19] Yvan Saeys, Iñaki Inza, and Pedro Larrañaga. A review of feature selection techniques in bioinformatics. *bioinformatics*, 23(19):2507–2517, 2007.

[20] Roberto Valenti, Nicu Sebe, Theo Gevers, and Ira Cohen. Machine learning techniques for face analysis. In Matthieu Cord and Pádraig Cunningham, editors, *Machine Learning Techniques for Multimedia*, Cognitive Technologies, pages 159–187. Springer Berlin Heidelberg, 2008.

[21] Giorgio Valentini and Thomas G Dietterich. Bias-variance analysis of support vector machines for the development of svm-based ensemble methods. *The Journal of Machine Learning Research*, 5:725–775, 2004.

[22] Jake VanderPlas. Frequentism and bayesianism: a python-driven primer. *arXiv preprint arXiv:1411.5018*, 2014.