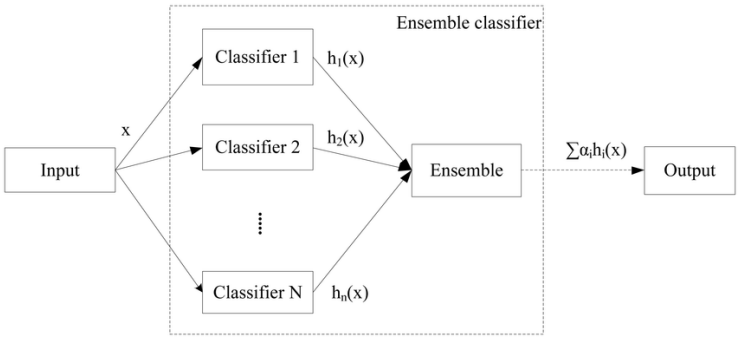
Ensemble Learning

Aarav Khanna

January 2020

1 Introduction

Ensemble Learning in Machine Learning combines the decisions from multiple models to improve the overall performance. This technique is primarily used to improve the (classification, prediction, function approximation, etc.) performance of a model, or reduce the likelihood of an unfortunate selection of a poor one. This approach allows for the production of better predictive performance compared to a single model.



1.1 Uses

Ensemble models have performed extremely well in many prestigious machine learning competitions. The Netflix Prize competition was a $1 Million competition to develop the best algorithm to predict user ratings for films. The winners used an ensemble model and even beat Netflix's algorithm. Furthermore, The KDD cup is an annual Data Mining competition, and in 2009, the winners used an Ensemble model as well.

2 Advantages

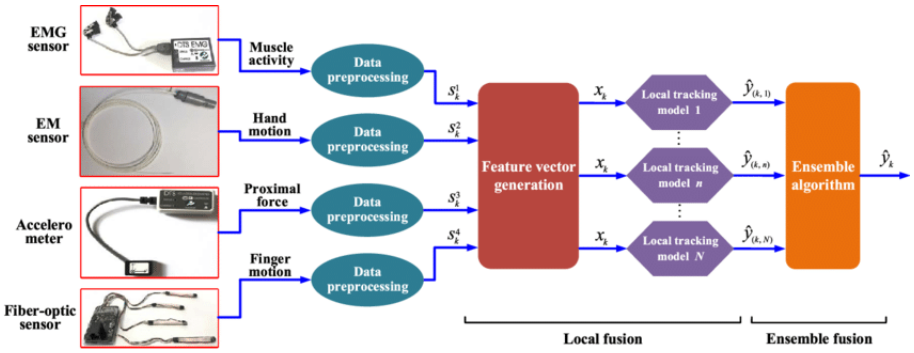
2.1 Too much or too little data

Ensemble based systems can be useful when dealing with large volumes of data or the lack of adequate data. When the amount of training data is too large

to make a single classifier training difficult, the data can be strategically partitioned into smaller subsets. Each partition can then be used to train a separate classifier which can then be combined using an appropriate combination rule. If, on the other hand, there is too little data, then bootstrapping can be used to train different classifiers using different bootstrap samples of the data, where each bootstrap sample is a random sample of the data drawn with replacement and treated as if it was independently drawn from the underlying distribution. We will discuss bootstrapping further later in the lecture.

2.2 Data Fusion

In many applications that call for automated decision making, it is not unusual to receive data obtained from different sources that may provide complementary information. A suitable combination of such information is known as data or information fusion, and can lead to improved accuracy of the classification decision compared to a decision based on any of the individual data sources alone. For example, for diagnosis of a neurological disorder, a neurologist may use the electroencephalogram (one-dimensional time series data), magnetic resonance imaging MRI, functional MRI, or positron emission tomography PET scan images (two-dimensional spatial data), the amount of certain chemicals in the cerebrospinal fluid along with the subjects demographics such as age, gender, education level of the subject, etc. These heterogeneous features cannot be used all together to train a single classifier (and even if they could - by converting all features into a vector of scalar values - such a training is unlikely to be successful). In such cases, an ensemble of classifiers can be used (Parikh 2007), where a separate classifier is trained on each of the feature sets independently.



2.3 Confidence Estimation

The very structure of an ensemble based system naturally allows assigning a confidence to the decision made by such a system. Consider having an ensemble of classifiers trained on a classification problem. If a vast majority of the classifiers agree with their decisions, such an outcome can be interpreted as the ensemble having high confidence in its decision. If, however, half the classifiers make one

decision and the other half make a different decision, this can be interpreted as the ensemble having low confidence in its decision. It should be noted that an ensemble having high confidence in its decision does not mean that decision is correct, and conversely, a decision made with low confidence does not mean it's incorrect. However, it has been shown that a properly trained ensemble decision is usually correct if its confidence is high, and usually incorrect if its confidence is low. Using such an approach then, the ensemble decisions can be used to estimate the probabilities of the classification decisions (Muhlbaier 2005).

3 Error

Before we discuss different Ensemble techniques, we must understand prediction errors (noise, bias, and variance). There is a trade off between a model's ability to minimize bias and variance. Gaining a proper understanding of these errors would help us not only to build accurate models but also to avoid the mistake of overfitting and underfitting. Let's start with the basics and see how they make a difference to our models.

3.1 Noise

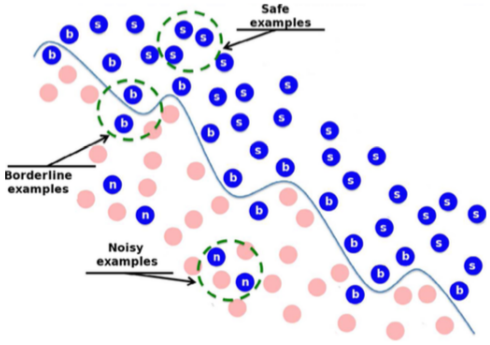
It's important to understand noise since it's a fundamental concept that can occur in every dataset. Lets begin with a simple understanding of noise. Noise is an unwanted distortion in data. It is anything that is spurious and extraneous to the original data, that is not intended to be present in the first place, but is introduced due to faulty capturing process. Noise can occur in many ways:

- If the data is an image, then the image noise in data can be due to poor lighting during capturing the image. It can also be due to the sensor (camera) which is of low quality, not able to capture high density picture.
- If the data is text, then the textual noise can be due to spelling errors, typography, excessive use of informal language, text which does not have semantic coherence, garbled text during text capture (OCR) etc..
- The same can be applicable to any other forms of data capture such as Video or Audio or any other forms like x-ray or spectral capture. Most of the noise happens due to noise in the nature (environment is not a perfect vacuum) or noise in the sensor (the equipment is not high quality).
- Noise can be introduced during storage, transfer or input/output processing as well.

We always must account for noise in Machine Learning. Now, let's say we have some data that we are using to train our Neural Net models. Let's say that the input vectors x1, x2,... xn has outcomes y1, y2...yn associated with it. Then the way to think about Noise is as follows:

$$y = f(x) + \epsilon$$

Where f(x) is a function on the independent variable x (input features) and y is the outcome. Here, epsilon is the noise in the data that has a functional relation with the input, and has a normal distribution with a mean of 0. The objective of the network is to learn the function f(x) to predict the outcomes. As you may guess, since there is a noisy functional relation (epsilon) between the function f(x) and the outcome, the probability of the model to learn about the noise is high.



3.2 Bias

Bias is the inability of the Neural Networks to learn any relation between the input vector and the output state. In simpler terms, it can be defined as how inaccurate our model is. Mathematically, it can be expressed as follows:

$$Bias[\hat{f}(x)] =< \hat{f}(x) - f(x) >$$

If the expected value of the difference between the learnt function and the underlying function is high, then we state that the learnt function is highly-biased.

3.3 Variance

Variance is the amount that the estimate of the target function will change if different training data was used. The target function is estimated from the training data by a machine learning algorithm, so we should expect the algorithm to have some variance. Ideally, it should not change too much from one training dataset to the next, meaning that the algorithm is good at picking out the hidden underlying mapping between the inputs and the output variables. Machine learning algorithms that have a high variance are strongly influenced by the specifics of the training data. This means that the specifics of the training have influences on the number and types of parameters used to characterize the mapping function.

- Low Variance: Suggests small changes to the estimate of the target function with changes to the training dataset.

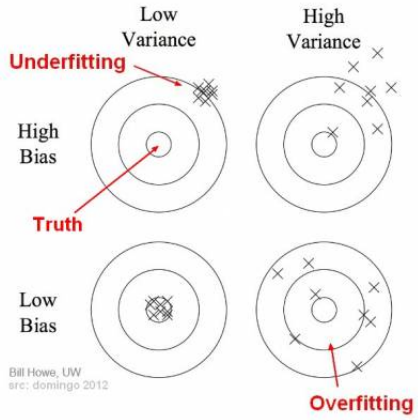
- High Variance: Suggests large changes to the estimate of the target function with changes to the training dataset.

Variance can also be explained in another way. Variance can be described as the sensitivity of the model to small changes in the inputs. In other words, any minor noise in the input gets picked up by the learning functions of the model and tries to over-fit the noise as if they are signal. This causes over-fitting and produces poor accuracy during validation. Mathematically variance can be expressed as follows:

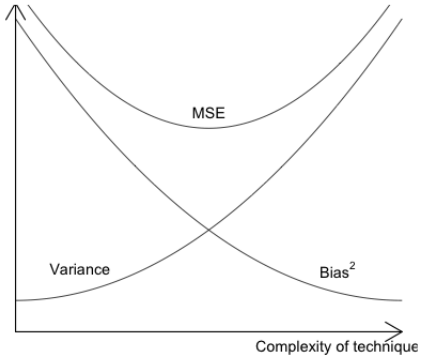
$$Var[\hat{f}(x)] =< \hat{f}(x)^2 > - < \hat{f}(x) >^2$$

3.4 Bias-Variance Trade-Off

Before we talk about the total error of a function, we must understand the Bias-Variance trade off. Most models do not have both a low bias and variance, so we must find the perfect balance between bias and error for minimal error. High bias, low variance algorithms train models that are consistent, but normally inaccurate. High variance, low bias algorithms train models that are normally accurate, but inconsistent.



But why is there a Bias-Variance trade off? If our model is too simple and has very few parameters then it may have high bias and low variance. On the other hand if our model has large number of parameters then it's going to have high variance and low bias. So we need to find the right balance without overfitting and underfitting the data. This tradeoff in complexity is why there is a tradeoff between bias and variance. An algorithm can't be more complex and less complex at the same time.



3.5 Total error

To build a good model, we need to find a good balance between bias and variance such that it minimizes the total error. Error can be calculated as:

$$Error(x) = Bias^2 + Variance + IrreducibleError$$

$$Error(x) = (E[\hat{f}(x)] - f(x))^2 + E[(\hat{f}(x) - E[\hat{f}(x)])^2] + \sigma^2$$

4 Ensemble Methods

Ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model in order to decrease variance (bagging), bias (boosting), or improve predictions (stacking). Ensemble methods can be divided into two groups:

4.1 Sequential Ensemble Methods

Sequential ensemble methods: These methods take advantage of the dependence between the base learners. Just like the classroom of students discussing a problem and getting the solution by complementing on the answers of the other students, the base learners complement the results of their base learners. A learner suggests a prediction which is way off from the correct prediction, another learner uses the suggested prediction by correcting it and suggesting another prediction with less error, then another learner uses the previous suggested prediction corrects it and suggests another prediction closer to the correct prediction while reducing the error and on and on. The basic motivation of sequential methods is to exploit the dependence between the base learners. The overall performance can be boosted by weighing previously mislabeled examples with higher weight. These methods are commonly known as boosting methods (e.g. AdaBoost).

4.2 Parallel Ensemble Methods

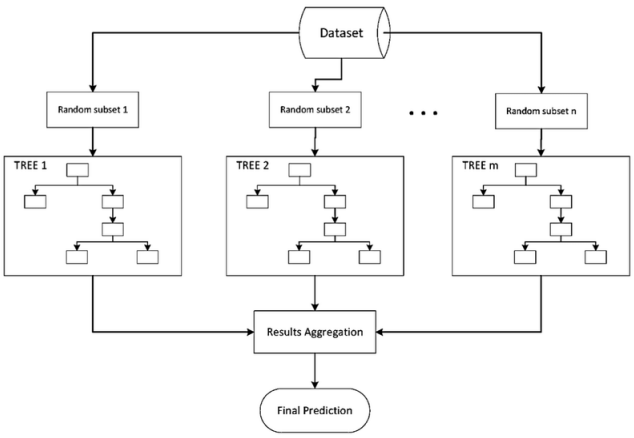
Parallel ensemble methods: These methods take advantage of the independence between the base learners. These methods take a path of democracy. Some algorithms allow the base learners to make predictions then picks the prediction which has the highest vote. Those are commonly known as voting methods. While other algorithms build multiple base learners from different samples of the training set to make predictions. The basic motivation of parallel methods is to exploit independence between the base learners since the error can be reduced dramatically by averaging. These are known as bagging methods (e.g. Random Forests).

5 Ensemble Techniques

Now we will discuss three basic ensemble techniques. We will discuss bagging (parallel), boosting (sequential), and stacking.

5.1 Bagging (Bootstrap Aggregating)

Bagging is an ensemble method. First, we create random samples of the training data set with replacement (sub sets of training data set). Then, we build a model for each sample. Finally, results of these multiple models are combined using average or majority voting. As each model is exposed to a different subset of data and we use their collective output at the end, so we are making sure that problem of overfitting is taken care of by not clinging too closely to our training data set. Thus, Bagging helps us to reduce the variance error. Combinations of multiple models decreases variance, especially in the case of unstable models, and may produce a more reliable prediction than a single model. Random forest technique actually uses this concept but it goes a step ahead to further reduce the variance by randomly choosing a subset of features as well for each bootstrapped sample to make the splits while training.



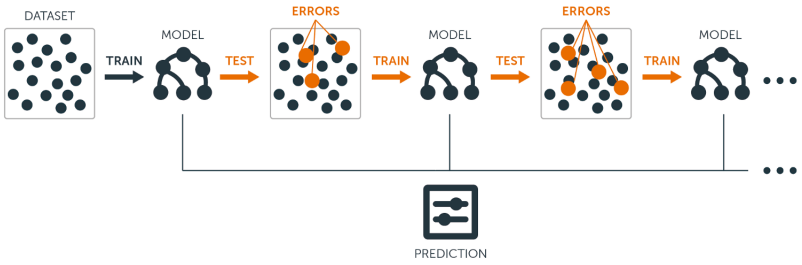
For example, we can train M different trees on different subsets of the data (chosen randomly with replacement) and compute the ensemble:

$$f(x) = 1/M \sum_{m=1}^{M} f_m(x)$$

For aggregating the outputs of base learners, bagging uses voting for classification and averaging for regression. The main purpose of bagging is to decrease the model's variance.

5.2 Boosting

Boosting is an iterative technique which adjusts the weight of an observation based on the last classification. If an observation was classified incorrectly, it tries to increase the weight of this observation and vice versa. The first algorithm is trained on the entire data set and the subsequent algorithms are built by fitting the residuals of the first algorithm, thus giving higher weight to those observations that were poorly predicted by the previous model. It relies on creating a series of weak learners, each of which might not be good for the entire data set but is good for some part of the data set. Thus, each model actually boosts the performance of the ensemble. Boosting in general decreases the bias error and builds strong predictive models. Boosting has shown better predictive accuracy than bagging, but it also tends to over-fit the training data as well. Thus, parameter tuning becomes a crucial part of boosting algorithms to make them avoid overfitting. The main purpose of boosting is to decrease the model's bias.



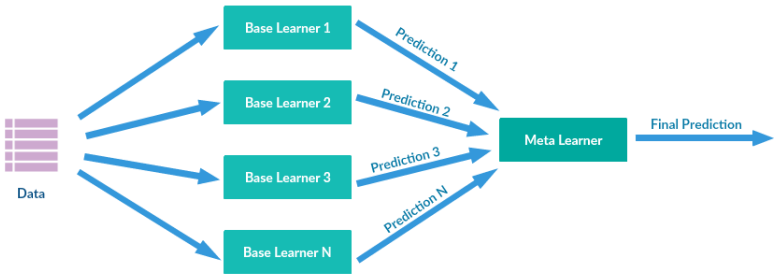
The algorithm below describes the most widely used form of boosting algorithm called AdaBoost, which stands for adaptive boosting.

Algorithm AdaBoost

1: Init data weights $\{w_n\}$ to $1/N$
2: **for** $m = 1$ to M **do**
3: fit a classifier $y_m(x)$ by minimizing weighted error function J_m:
4: $J_m = \sum_{n=1}^{N} w_n^{(m)} 1[y_m(x_n) \neq t_n]$
5: compute $\epsilon_m = \sum_{n=1}^{N} w_n^{(m)} 1[y_m(x_n) \neq t_n] / \sum_{n=1}^{N} w_n^{(m)}$
6: evaluate $\alpha_m = \log\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$
7: update the data weights: $w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m 1[y_m(x_n) \neq t_n]\}$
8: **end for**
9: Make predictions using the final model: $Y_M(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(x)\right)$

5.3 Stacking

Stacking is an ensemble learning technique that combines multiple classification or regression models via a meta-classifier or a meta-regressor. The base level models are trained based on a complete training set, then the meta-model is trained on the outputs of the first model as features. The base level often consists of multiple different learning algorithms.



In simpler terms, stacking is another ensemble model where a new model is trained from the combined predictions of two (or more) previous model. The predictions from the previous models are used as inputs for each sequential layer, and combined to form a new set of predictions. These can be used on additional layers, or the process can stop here with a final result. The algorithm below summarizes stacking.

Algorithm Stacking

Input: training data $D = \{x_i, y_i\}_{i=1}^{m}$
Ouput: ensemble classifier H
Step 1: learn base-level classifiers
for $t = 1$ to T **do**
    learn h_t based on D
end for
Step 2: construct new data set of predictions
for $i = 1$ to m **do**
    $D_h = \{x'_i, y_i\}$, where $x'_i = \{h_1(x_i), ..., h_T(x_i)\}$
end for
Step 3: learn a meta-classifier
learn H based on D_h
return H

6 Disadvantages of Ensemble Methods

1. Reduction in model interpret ability- Using ensemble methods reduces the model interpret-ability due to increased complexity and makes it very difficult to draw any crucial insights about the model in the end.

2. Computation and design time is high- It is not good for real time applications.

3. The selection of models for creating an ensemble is an art which is really hard to master.

4. The model that is closest to the true data generating process will normally be the best and will beat most ensemble methods. Thus, if the data come from a linear process, linear models will be much superior to ensemble models.

5. Ensemble methods are usually computationally expensive. Therefore, they add learning time and memory constrains to the problem.

7 Code

Below is an Ensemble code made using the Stacking method. It displays the accuracy of a KNN, Random Forest, Naive Bayes, and a Stacking algorithm (uses all three) on the Iris dataset.

```
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from mlxtend.classifier import StackingClassifier
```

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier

iris = datasets.load_iris()
X, y = iris.data[:, 1:3], iris.target

def CalculateAccuracy(y_test,pred_label):
    nnz = np.shape(y_test)[0] - np.count_nonzero(pred_label - y_test)
    acc = 100*nnz/float(np.shape(y_test)[0])
    return acc

clf1 = KNeighborsClassifier(n_neighbors=2)
clf2 = RandomForestClassifier(n_estimators = 2,random_state=1)
clf3 = GaussianNB()
lr = LogisticRegression()

clf1.fit(X, y)
clf2.fit(X, y)
clf3.fit(X, y)

f1 = clf1.predict(X)
acc1 = CalculateAccuracy(y, f1)
print("accuracy from KNN: "+str(acc1) )

f2 = clf2.predict(X)
acc2 = CalculateAccuracy(y, f2)
print("accuracy from Random Forest: "+str(acc2) )

f3 = clf3.predict(X)
acc3 = CalculateAccuracy(y, f3)
print("accuracy from Naive Bays: "+str(acc3) )

f = [f1,f2,f3]
f = np.transpose(f)

lr.fit(f, y)
final = lr.predict(f)

acc4 = CalculateAccuracy(y, final)
print("accuracy from Stacking: "+str(acc4) )

#Output:
#accuracy from KNN: 96.66666666666667
#accuracy from Random Forest: 94.66666666666667
#accuracy from Naive Bays: 92.0
#accuracy from Stacking: 97.33333333333333
```

8 Conclusion

The goal of any machine learning problem is to find a single model that will best predict our wanted outcome. Rather than making one model and hoping this model is the best/most accurate predictor we can make, ensemble methods take a myriad of models into account, and average those models to produce one final model. Ensemble Models have continued to be more relevant in data science, and it is essential that we understand both the advantages and drawbacks.

9 Recommended Reading

These sources provide more information on Ensemble Learning if you are interested:

- Zhi-Hua Zhou, "Ensemble Methods: Foundations and Algorithms", CRC Press, 2012
- L. Kuncheva, "Combining Pattern Classifiers: Methods and Algorithms", Wiley, 2004
- Kaggle Ensembling Guide
- Scikit Learn Ensemble Guide
- S. Rachka, MLxtend library

10 Acknowledgements

Here are the sources I used to make my lecture:

- https://medium.com/@rrfd/boosting-bagging-and-stacking-ensemble-methods-with-sklearn-and-mlens-a455c0c982de
- https://blog.statsbot.co/ensemble-learning-d1dcd548e936
- https://towardsdatascience.com/ensemble-methods-in-machine-learning-what-are-they-and-why-use-them-68ec3f9fef5f
- https://www.analyticsvidhya.com/blog/2015/08/introduction-ensemble-learning/
- http://www.scholarpedia.org/article/Ensemble_learning