Data Mining 2

Topic 05 : Ensemble Learning

Lecture 01 : Introduction to Ensemble Learning

Dr Kieran Murphy

Department of Computing and Mathematics, WIT.
(kmurphy@wit.ie)

Spring Semester, 2021

Outline

- Ensemble learners
- Bootstrapping and Bagging
- Boosting, AdBoost

Outline

Motivation

Condorcet's jury theorem is a political science theorem about the relative probability of a given group of individuals arriving at a correct decision:

Theorem 1 (Condorcet's jury theorem)

Assume a group of n *independent voters wishes to reach a decision by majority vote. One of the two outcomes of the vote is correct, and each voter has an independent probability,* p*, of voting for the correct decision.*

- *If* $p > 0.5$*, then adding more voters increases the probability that the majority decision is correct. In the limit, the probability that the majority votes correctly approaches 1 as the number of voters,* n *increases.*
- *If* $p < 0.5$ *then adding more voters makes things worse: the optimal jury consists of a single voter.*

Take Home Message

- Even weak decision makers have benefit as long as they individualy perform better than chance ($p > 0.5$).
- "Wisdom of Crowds" needs independence!
- What happens if $p < 0.5$?

Condorcet's Jury Theorem

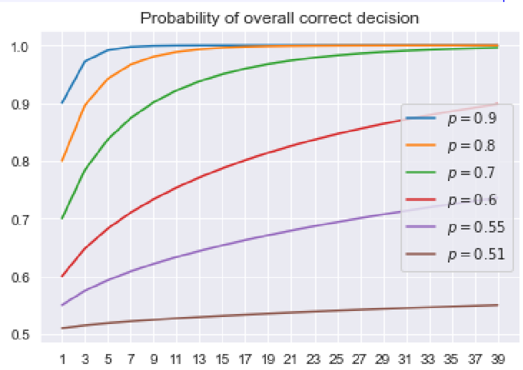
```python
from scipy.stats import binom
nValues = np.array(range(1,41,2))

prob_correct = lambda n,p: 1-binom.cdf(n//2, n, p)

for p in [0.9, 0.8, 0.7, 0.6, 0.55, 0.51]:
    muValues = prob_correct(nValues,p)
    plt.plot(nValues,muValues,label="$p=%s$" % p)

plt.title("Probability of overall correct decision")
plt.xticks(nValues)
plt.legend(loc="center right")
plt.savefig("jury_decision.pdf",bbox_inches="tight")
plt.show()
```

… just calculating cumulative binomial distribution probabilities.

Condorcet's Jury Theorem

```
from scipy.stats import binom
nValues = np.array(range(1,41,2))

prob_correct = lambda n,p: 1-binom.cdf(n//2, n, p)

for p in [0.9, 0.8, 0.7, 0.6, 0.55, 0.51]:
    muValues = prob_correct(nValues,p)
    plt.plot(nValues,muValues,label="$p=%s$" % p)

plt.title("Probability of overall correct decision"
plt.xticks(nValues)
plt.legend(loc="center right")
plt.savefig("jury_decision.pdf",bbox_inches="tigh
plt.show()
```

... just calculating cumulative binomial distribution probabilities.
With enough voters the probability of correct decision approaches one.
'enough' gets large for $p \approx 0.5$.



Practical Motivation — Netflix Prize*

- Open competition to predict user ratings for films, based only on previous ratings.
- Prize was awarded on 2009 to BellKor's Pragmatic Chaos team which bested Netflix's own algorithm for predicting ratings by 10.06%. was an ensemble of 107 modules.

Leaderboard

Display top 20 leaders.

Rank	Team Name	Best Score	% Improvement	Last Submit Time
1	The Ensemble	0.8553	10.10	2009-07-26 18:38:22
2	BellKor's Pragmatic Chaos	0.8554	10.09	2009-07-26 18:18:28
Grand Prize - RMSE <= 0.8563				
3	Grand Prize Team	0.8571	9.91	2009-07-24 13:07:49
4	Opera Solutions and Vandelay United	0.8573	9.89	2009-07-25 20:05:52
5	Vandelay Industries !	0.8579	9.83	2009-07-26 02:49:53
6	PragmaticTheory	0.8582	9.80	2009-07-12 15:09:53
7	BellKor in BigChaos	0.8590	9.71	2009-07-26 12:57:25
8	Dace_	0.8603	9.58	2009-07-24 17:18:43
9	Opera Solutions	0.8611	9.49	2009-07-26 18:02:08
10	BellKor	0.8612	9.48	2009-07-26 17:19:11
11	BigChaos	0.8613	9.47	2009-06-23 23:06:52
12	Feeds2	0.8613	9.47	2009-07-24 20:06:46
Progress Prize 2008 - RMSE = 0.8616 - Winning Team: BellKor in BigChaos				
13	xiangliang	0.8633	9.26	2009-07-21 02:04:40
14	Gravity	0.8634	9.25	2009-07-26 15:58:34
15	Ces	0.8642	9.17	2009-07-25 17:42:38
16	Invisible Ideas	0.8644	9.14	2009-07-20 03:26:12
17	Just a guy in a garage	0.8650	9.08	2009-07-22 14:10:42
18	Craig Carmichael	0.8656	9.02	2009-07-25 16:00:54
19	J Dennis Su	0.8658	9.00	2009-03-11 09:41:54
20	acmehill	0.8659	8.99	2009-04-16 06:29:35
Progress Prize 2007 - RMSE = 0.8712 - Winning Team: KorBell				
Cinematch score on quiz subset - RMSE = 0.9514				

Ensembe Methods

Definition 2 (Ensemble Learner)

An ensemble learner is a set of models whose individual decisions are combined in some way to classify new examples.

- Simplest approach:
  - Generate multiple classifiers
  - Each votes on test instance
  - Take majority as classification
- Classifiers are different due to different sampling of training data, or randomised parameters within the classification algorithm.
- Aim: take simple mediocre algorithm and transform it into a super classifier without requiring any fancy new algorithm.
- Differ in training strategy, and combination method:
  - Parallel training with different training sets: Bagging or Cross-validated committees
  - Sequential training, iteratively re-weighting training examples so current classifier focuses on hard examples: boosting
  - Parallel training with objective encouraging division of labor: mixture of experts

Why do Ensemble Methods Work?

Variance reduction

If the training sets are completely independent, it will always help to average an ensemble because this will reduce variance without affecting bias (e.g., bagging)

- Reduce sensitivity to individual data points.

Bias reduction

For simple models, average of models has much greater capacity than single model (e.g., hyperplane classifiers, Gaussian densities).

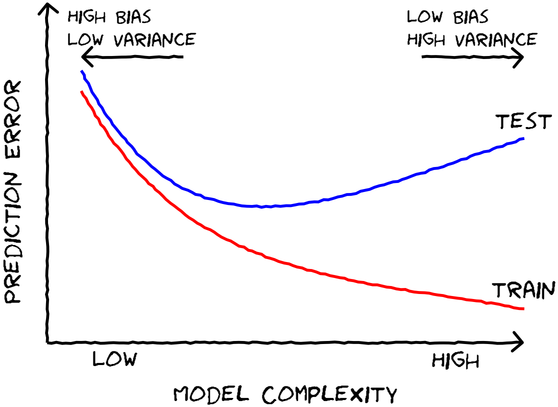
- Averaging models can reduce bias substantially by increasing capacity, and control variance by fitting one component at a time (e.g., boosting)

Classification vs Regression

- Regression models can be averaged.
- Classification models can be averaged if output is probability of being in a class, otherwise can use majority vote if classifier only outputs class.

Bias/Variance Tradeoff

Bias: error from erroneous assumptions in the model



Variance: error from sensitivity to small fluctuations in the training set

- Model too simple then has large bias (as model is too simple to learn signal) but small variance (as model is too simple to learn/be affected by noise).
- Model too complicated then has small bias (as model can learn signal) but has large variance (as model also can learn noise).

Reduce Variance Without Increasing Bias

It seems that all we can do to is select how complicated our model is to minimise the generalisation error ($\text{bias}^2 + \text{Var} + \text{noise}$). But this is not the case:

- It is possible to reduce variance without affecting bias by averaging.

Averaging reduces variance

- Given N independent estimates for X, each with variance of $\text{Var}(X)$, we have

$$\text{Var}(\bar{X}) = \frac{\text{Var}(X)}{N}$$

But only if independent!

One Problem

We have only one training set, so where do multiple models (independent estimates) come from? — apply Bootstrap sampling

Bootstrap sampling

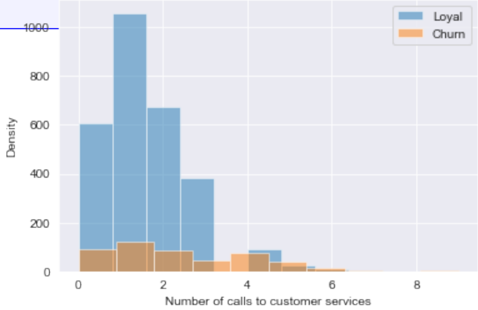
Given set D containing N training examples, create D' by drawing N examples at random with replacement from D.

Example — Bootstrapping for Small Samples I

Consider our Churn dataset with only 3333 rows. Lets look at the distribution of `Cust_Serv_Calls` for both the loyal customers and the churning customers . . .

```
df.loc[df['Churn']==0,'Cust_Serv_Calls'].hist(label='Loyal',alpha=0.5)
df.loc[df['Churn']==1,'Cust_Serv_Calls'].hist(label='Churn',alpha=0.5)
plt.xlabel('Number of calls to customer services')
plt.ylabel('Density')
plt.legend()
plt.savefig("churn__Cust_Serv_Calls__hist.pdf",bbox="tight")
plt.show()
```

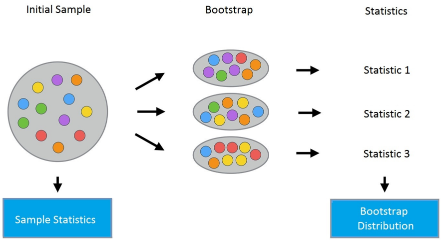
- Looks like loyal customers make fewer calls to customer service than those who eventually leave.
- So we should estimate the average number of customer service calls in each group.
- As dataset is small, we would not get a good estimate by simply calculating the mean of the original samples.
- We would be better off applying the bootstrap method . . .



Example — Bootstrapping for Small Samples II

Create two utility functions to generate the bootstrap samples and to compute statistic estimates:

- We could have used `np.random.choice` with option `replace=True` to get the same effect.
- $\alpha\%$-confidence intervals are computed as done back in semester 2.




```python
def get_bootstrap_samples(data, n_samples):
    """Generate bootstrap samples using the bootstrap method."""
    indices = np.random.randint(0, len(data), (n_samples, len(data)))
    samples = data[indices]
    return samples

def stat_intervals(stat, alpha):
    """Produce an interval estimate."""
    boundaries = np.percentile(stat, [100*alpha/2, 100*(1-alpha/2)])
    return boundaries
```

Example — Bootstrapping for Small Samples III

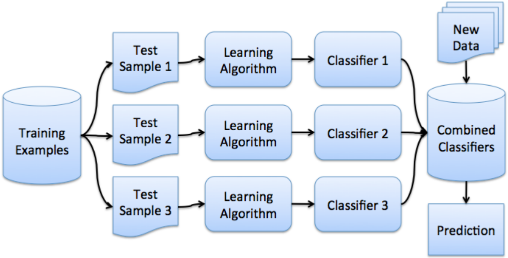
Next we split the data set, generate bootstrap samples and resulting confidence intervals . . .

```
np.random.seed(42)

loyal_calls = df.loc[df['Churn']==0, 'Cust_Serv_Calls'].values
churn_calls = df.loc[df['Churn']==1, 'Cust_Serv_Calls'].values

# Generate the samples using bootstrapping and calculate the mean
loyal_mean_scores = [np.mean(sample)
    for sample in get_bootstrap_samples(loyal_calls, 1000)]
churn_mean_scores = [np.mean(sample)
    for sample in get_bootstrap_samples(churn_calls, 1000)]

print("Service calls from loyal: mean interval",
    stat_intervals(loyal_mean_scores, 0.05))
print("Service calls from churn: mean interval",
    stat_intervals(churn_mean_scores, 0.05))
```

```
Service calls from loyal: mean interval [1.40700877 1.4922807 ]
Service calls from churn: mean interval [2.06625259 2.38307453]
```

Outline

Bagging aka Bootstrap AGgregation



Method

- Create M bootstrap samples, $D_1, D_2, \ldots, D_M$ of size N.
  - When picking each element of sample D_i, each element in D has probability of $1/N$ of being selected.
  - The probability of an element of D not being selected for D_i is

$$(1 - 1/N)^N \xrightarrow[N \to \infty]{} 1/e$$

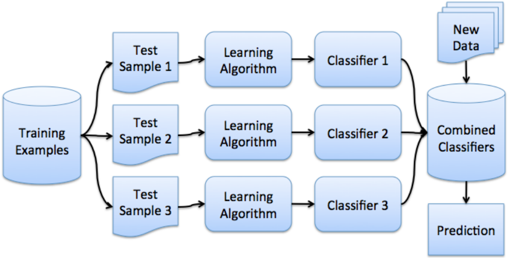
  - Hence the probability of an element of D being selected for D_i is $1 - 1/e = 0.632$.

A bootstrap sample contains 63% of the original data.

- Separately train classifier on each D_i.
- Classify new instance by majority vote / average.

*Leo Breiman (1994)

Bagging — aka Bootstrap AGgregation



Method

- Create M bootstrap samples, $D_1, D_2, \ldots, D_M$ of size N.
  - When picking each element of sample D_i, each element in D has probability of $1/N$ of being selected.
  - The probability of an element of D not being selected for D_i is

$$\left(1 - 1/N\right)^N \xrightarrow[N \to \infty]{} 1/e$$

  - Hence the probability of an element of D being selected for D_i is $1 - 1/e = 0.632$.

A bootstrap sample contains 63% of the original data.

- Separately train classifier on each D_i.
- Classify new instance by majority vote / average.

*Leo Breiman (1994)

Bagging — Performance I

Definition 3 (Unstable learner)

A learner is unstable if its output classifier undergoes major changes in response to small changes in training data

- Unstable: decision-tree, neural network, rule learning algorithms, ...
- Stable: linear regression, nearest neighbour, linear threshold algorithms, ...

Bagging tends to

- works well for unstable learners
- can have a mild negative effect on the performance of stable methods

Best Case

$$\mathrm{Var}\Big(\mathrm{Bagging}\big(L(x,D)\big)\Big) = \frac{\mathrm{Var}\big(L(x,D)\big)}{M}$$

However, in practice the models are correlated, so reduction is smaller than $1/M$. Also variance of models trained on fewer training cases can be somewhat larger.

Bagging — Performance II

- Bagging reduces the variance of a classifier by decreasing the difference in error when we train the model on different datasets.
- In other words, bagging prevents overfitting.
- The efficiency of bagging comes from the fact that the individual models are quite different due to the different training data and their errors cancel each other out during voting.
- Additionally, outliers are likely omitted in some of the training bootstrap samples.
- Bagging is effective on small datasets.
  - Dropping even a small part of training data leads to constructing substantially different base classifiers.
  - If you have a large dataset, you would generate bootstrap samples of a much smaller size.

Example

The skikit-learn documentation has a simulation showing the effect of bagging.

Outline

Motivation — ‘How May I Help You?’

Problem

Automatically categorise type of call requested by phone customer (`Collect`, `CallingCard`, `PersonToPerson`, etc.)

- ‘Yes I’d like to place a collect call long distance please’ (`Collect`)
- ‘Operator I need to make a call but I need to bill it to my office’ (`ThirdNumber`)
- ‘Yes I’d like to place a call on my master card please’ (`CallingCard`)
- ‘I just called a number in sioux city and I musta rang the wrong number because I got the wrong party and I would like to have that taken off of my bill’ (`BillingCredit`)

Observations

- Easy to find ‘rules of thumb’ that are ‘often’ correct
  e.g. ‘IF "card" occurs in utterance THEN predict ‘`CallingCard`’
- Hard to find single highly accurate prediction rule.

---

*Gorin et al

Boosting Approach

Outline

- Devise procedure for deriving rough rules of thumb
- Apply procedure to subset of examples
- Obtain rule of thumb
- Apply to 2nd subset of examples
- Obtain 2nd rule of thumb
- Repeat T times

Key Steps

- How do we choose examples on each round?
  - Concentrate on hardest examples (those most often miss-classified by previous rules of thumb) — focus by sampling or weighing the whole data set.
- How do we combine rules of thumb into a single prediction rule?
  - Take (weighted) majority vote of rules of thumb.

If Boosting is possible, then

- can use (fairly) wild guesses to produce highly accurate predictions.
- for any learning problem:
  - either can always learn with nearly perfect accuracy
  - or there exist cases where cannot learn even slightly better than random guessing

Boosting

Boosting

General method of converting rough rules of thumb into highly accurate prediction rule.

- Assume given 'weak' learning algorithm that can consistently find classifiers ('rules of thumb') at least slightly better than random, say, accuracy $\geq 55\%$ (in two-class setting).

  'weak learning assumption'

- Then given sufficient data, a boosting algorithm can provably construct single classifier with very high accuracy, say, 99%.
- In contrast to bagging which has little effect on Bias, boosting reduces both bias and variance.

History

Schapire '89 — first provable boosting algorithm

Freund '90 — 'optimal' algorithm that 'boosts by majority'

nd & Schapire '95 — introduced 'AdaBoost' algorithm, strong practical advantages over previous boosting algorithms

AdaBoost Algorithm I

Outline

- First train the base classifier on all the training data with equal importance weights on each case.

  Given weights and data how do we train a classifier?

- Then re-weight the training data to emphasise the hard cases and train a second model.

  How do we re-weight the data?

- Repeat above steps.
- Finally, use a weighted committee of all the models for the test data.

  How do we weight the models in the committee?

Notation

- Input: feature matrix, X; target vector, $\boldsymbol{y} \in \{-1, 1\}$
  - Recall: X_m is m(th) feature/column and X^n is n(th) row/example — superscript represent rows/examples/cases.
- Output: m(th) classifier predicted output $\boldsymbol{f}_m(X) = \{-1, 1\}$

AdaBoost Algorithm — How do we train a classifier?

Weights

Let $\boldsymbol{w}_m^n$ represent the weights of example n for classifier m. With

$$\boldsymbol{w}_1^n = 1/N$$

Whenever, we change these weights, we will rescale so that they sum to one.

Training

Given feature matrix, X, target vector, $\boldsymbol{y}$ and weights w_m^n, we train a (weak) classifier using cost function for classifier m:

$$J_m = \sum_{n=1}^{N} w_m^n \underbrace{\left[f_m(X^n) \neq y^n\right]}_{\text{1 if error, else 0}} = \sum \text{weighted errors}$$

AdaBoost Algorithm — How do we update weights?

- Define the unnormalized error rate of a classifier as

$$\epsilon_m = J_m$$

and the quality of the classifier as

$$\alpha_m = \frac{1}{2} \ln \left(\frac{1 - \epsilon_m}{\epsilon_m} \right)$$

This is zero if the classifier has weighted error rate of 0.5 and infinity if the classifier is perfect.

- The weights for the next round are then

$$w^n_{m+1} = w^n_m \cdot \frac{\exp\{-\alpha_m y^n f_m(X^n)\}}{\sum_{n=1}^N w^n_m \exp\{-\alpha_m y^n f_m(X^n)\}}$$

Notice the product $y^n f_m(X^n)$.

  - This is +1 when prediction matches actual and -1 otherwise.
  - So $\exp\{-\alpha_m y^n f_m(X^n)\}$ will be small when prediction matches actual and big otherwise $\implies$ increases weight for harder cases to focus on them.

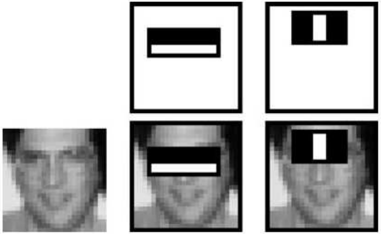
AdaBoost Algorithm — How do we make predictions?

After M boosting iterations, we have m classifiers. To use in prediction of new cases we weight the binary prediction of each classifier by the quality of that classifier:

$$f(X_{\text{test}}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m f_m(X_{\text{test}})\right)$$

Example Application of boosting

- Viola and Jones created a very fast face detector that can be scanned across a large image to find the faces.



- Two twists on standard algorithm:
  - Pre-define weak classifiers, so optimisation=selection
    - The base classifier/weak learner just compares the total intensity in two rectangular pieces of the image — very fast operation.
  - Change loss function for weak learners: false positives less costly than misses