Turning Lambs into Lions – Winning at Machine Learning using Ensemble Methods

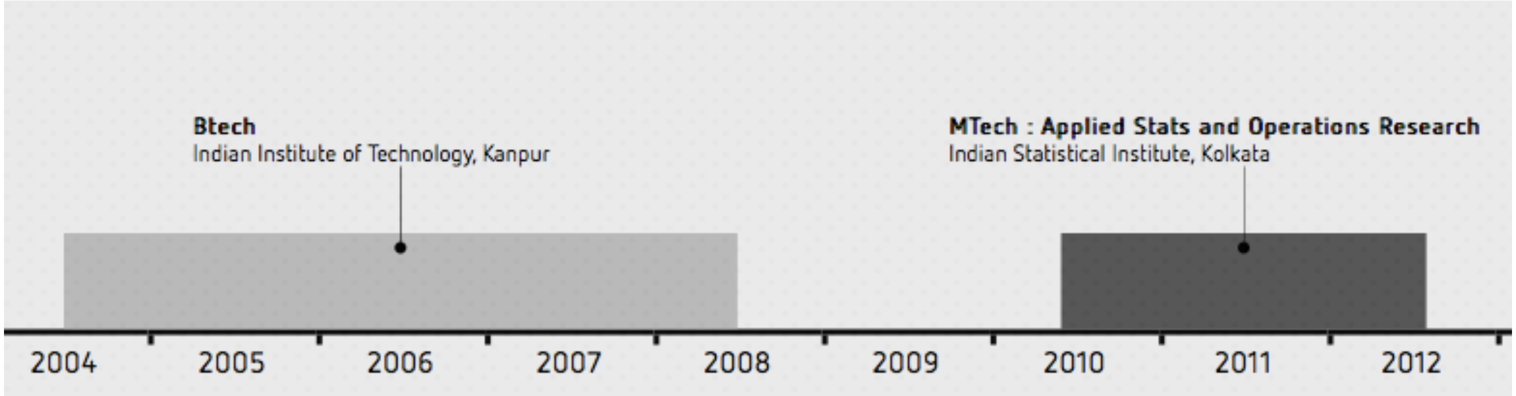
EDVANCER
EDUVENTURES

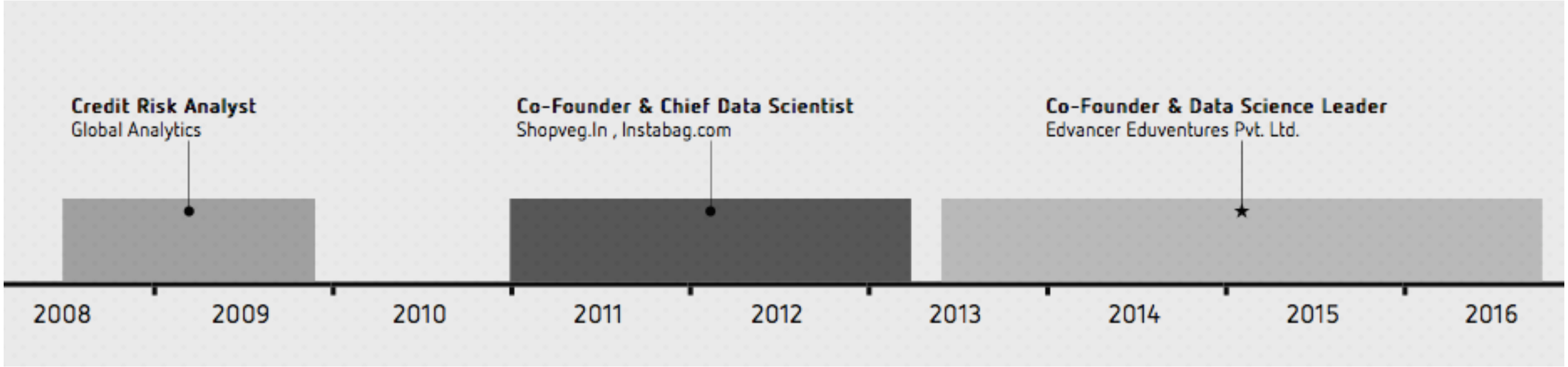
About Edvancer

- In last 3+ years , we have helped thousands of independent individuals and 100+ corporate clients towards building their capabilities in data science
- Here are few of our clients :

About Me

Lalit Sachan
eMail : lalit.sachan@edvancer.in
Linkedin : https://in.linkedin.com/in/lalitsachan





Preview

Contd..

- one line summary for the talk :How to improve performance of existing traditional models by combining them
- Flow :
  - Brief note on individual algos
  - Simple majority vote with example
  - blending
  - stacking
  - Ideas on further possibilities
  - Summary
  - Q&A

What we already do

Existing Models

Logistic Regression	Linear Decision Boundary [without variable transformation]
SVM	Non- Linear Decision Boundaries [Depends on choice of kernels]
Other Algos	They tend to capture some specific trends depending on choice of hyper-parameters and data subset

Contd..

- Take Away :different methods capture slightly different patterns depending on choice of hyper parameters and their own inherent nature

- This holds for both regression and classification

- Goal of ensembles is to utilise these differences to capture different trends in a single model

Let them have their say

EDVANCER EDUVENTURES

A simple democracy of models

Real outcome	1	1	1	1	1	1	1	1	1	1
Consider 3 models with 70% accuracy	1	0	0	1	0	1	1	1	1	1
	0	1	1	0	1	1	1	0	1	1
	1	1	1	0	1	0	1	1	1	0
Majority Vote : 90% Accuracy	1	1	1	0	1	1	1	1	1	1

A Little (over)generalisation

- Consider 3 models with 70% accuracy and think about these 2 scenarios
- P(All 3 outcomes are correct): 0.7 * 0.7 * 0.7 = 0.343
- P(2 are correct) = 3 * 0.7 * 0.7 * 0.3 = 0.441
- This gives us an overall accuracy ~ 78%
- Why is this better than the truth ?

The Reality

- All models will not be independent in reality

Consider 3 models with 70% accuracy but high correlation

1	*1*	*0*	*1*	*0*	*1*	*1*	*1*	*1*	*0*

1	*1*	*1*	*0*	*1*	*0*	*1*	*1*	*1*	*0*

1	*1*	*1*	*1*	*0*	*0*	*1*	*1*	*1*	*0*

1	*1*	*1*	*1*	*0*	*0*	*1*	*1*	*1*	*0*

- This does not improve accuracy at all !

Cautions

- Ensemble models which are less correlated . Simple pearson correlation can be used

- Ensemble need not be democratic .

  - A strong model can be given high weightage

  - A correction will happen if many weak models disagree with the strong one

  - This will not result in much improvement but might correct an already strong model

Case Study

Example : Majority Vote

- We are going to build several models on marketing campaign data where we'll be trying to model whether a customer will subscribe to certain campaign or not

- Predictors :
  - Personal Information : age , marital status …
  - Financial Indicators : balance , loan..
  - Previous Campaign : poutcome, pdays…

Data glimpse

```
> glimpse(d)
Observations: 45,211
Variables: 20
$ age            (dbl) 0.51948052, 0.33766234, 0.19480519, 0.37662338, 0.19480519,...
$ education      (dbl) 1.0000000, 0.6666667, 0.6666667, 0.0000000, 0.0000000, 1.00...
$ default        (dbl) 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
$ balance        (dbl) 0.09225936, 0.07306666, 0.07282153, 0.08647613, 0.07281245,...
$ housing        (dbl) 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
$ loan           (dbl) 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
$ day            (dbl) 0.1333333, 0.1333333, 0.1333333, 0.1333333, 0.1333333, 0.13...
$ month          (dbl) 0.3636364, 0.3636364, 0.3636364, 0.3636364, 0.3636364, 0.36...
$ duration       (dbl) 0.053070354, 0.030703538, 0.015453436, 0.018706791, 0.04026...
$ campaign       (dbl) 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000,...
$ pdays          (dbl) 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
$ previous       (dbl) 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
$ y              (dbl) 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
$ marital_div    (dbl) 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
$ marital_married (dbl) 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1,...
$ cont_cell      (dbl) 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
$ cont_unknown   (dbl) 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
$ pout_fail      (dbl) 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
$ pout_other     (dbl) 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
$ pout_success   (dbl) 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
```

contd.

```
set.seed(3)
s=sample(1:nrow(d),0.8*nrow(d))
d_train=d[s,]
d_test=d[-s,]


### KNN
knn_pred=class::knn(d_train[,-13],d_test[,-13],d_train[,13],k=50)
table(knn_pred,d_test[,13])
## GBM
set.seed(3)
gbm.fit=gbm::gbm(y~.,
            data=d_train,
            distribution = "bernoulli",
            n.trees = 2000,interaction.depth = 3,
            n.minobsinnode = 5,
            shrinkage = 0.05,
            verbose=T,
            n.cores=3)
k=gbm::gbm.perf(gbm.fit,method="OOB")
```

to tune : k for knn

to tune : for gbm
n.trees
interaction.depth
n.minobsinnode
shrinkage

Should use method “cv” here

contd.

```
# ET
ET.fit=extraTrees::extraTrees(d_train[,-13],as.factor(d_train$y))
et_pred=predict(ET.fit,d_test[,-13])
table(et_pred,d_test[,13])


fit_rf=randomForest::randomForest(factor(y)~.,do.trace=T,data=d_train)
fit_rf

rf_pred=predict(fit_rf,newdata=d_test)

# SVM
svm.fit=e1071::svm(d_train[,-13],d_train[,13],type="C-classification")
svm_pred=predict(svm.fit,d_test[,-13])

# logistic

log.fit=step(glm(y~.,data=d_train,family="binomial"))
log_pred=as.numeric(predict(log.fit,d_test,type="response")>0.2)
```

to tune :
for ET & RF :
ntrees
mtry

to tune : for svm : cost,kernel

Find threshold with a proper method

contd.

```
my_pred=data.frame(knn_pred=as.numeric(as.character(knn_pred)),
                    gbm_pred,
                   et_pred= as.numeric(as.character(et_pred)),
                   rf_pred=as.numeric(as.character(rf_pred)),
                   svm_pred=as.numeric(as.character(svm_pred)),
                   log_pred
                   )

cor(my_pred)
```

	knn_pred	gbm_pred	et_pred	rf_pred	svm_pred	log_pred
knn_pred	1.0000000	0.3685729	0.46630...	0.49304...	0.6906699	0.4169343
gbm_pred	0.3685729	1.0000000	0.60491...	0.58839...	0.4842839	0.7696112
et_pred	0.4663059	0.6049132	1.00000...	0.76974...	0.6242186	0.5978930
rf_pred	0.4930493	0.5883974	0.76974...	1.00000...	0.6909896	0.6114909
svm_pred	0.6906699	0.4842839	0.62421...	0.69098...	1.0000000	0.5542370
log_pred	0.4169343	0.7696112	0.59789...	0.61149...	0.5542370	1.0000000

contd.

```
max_freq=function(x){

  t=sort(table(x),decreasing = T)
  return(as.numeric(names(t)[1]))
}

my_pred$majority_vote=apply(my_pred,1,max_freq)
my_pred$w_majority_vote=apply(my_pred,1,function(x) max_freq(c(rep(x[2],4),x[-2]))
```

You can make weighted majority vote more sophisticated

Performance

```r
real=d_test[,13]
model_perf=data.frame(model="dummy",precision=99,recall=99,f1=99,acc=99)
for(i in 1:ncol(my_pred)){
  tp=sum(real==1 & my_pred[,i]==1)
  fp=sum(real==0 & my_pred[,i]==1)
  fn=sum(real==1 & my_pred[,i]==0)
  tn=sum(real==0 & my_pred[,i]==0)
  acc=(tp+tn)/(tp+tn+fp+fn)
  precision=tp/(tp+fp)
  recall=tp/(tp+fn)
  f1=2*(precision*recall)/(precision+recall)

  model_perf=rbind(model_perf,c(NA,precision,recall,f1,acc))
}

model_perf=model_perf[-1,]
model_perf$model=names(my_pred)
```

model	precision	recall	f1	acc
knn_pred	0.6778523	0.18481...	0.29043...	0.8908548
gbm_pred	0.5151699	0.77676...	0.61948...	0.8846622
et_pred	0.6216578	0.42543...	0.50516...	0.8992591
rf_pred	0.6630769	0.39432...	0.49454...	0.9025766
svm_pred	0.6652268	0.28179...	0.39588...	0.8960522
log_pred	0.4939759	0.60018...	0.54192...	0.8773637
majority_vote	0.6451078	0.35590...	0.45872...	0.8984850
w_majority_vote	0.6270338	0.45837...	0.52959...	0.9015813

Averaging

- Averaging can be used where outcome is continuos numeric
- Works with regression problems outcomes , probabilities in classification etc
- Ensembling less correlated models helps here as well
- Same goes for weighing different models differently while averaging

Beyond Bagging!

Blending

- Idea is to use model predictions as features

- Blending :

  - break data into 3 parts train_a, train_b, test
  - Build first layer models on train_a.
  - Make predictions on train_b , then use second layer model on train_b and test performance on the test data

Blending : Pros & Cons

- Simple and intuitive
- Easier to implement [in comparison to stacking]
- No information leakage

- Less data is used for second layer of model
- Second layer model might overfit the holdout data

Blending..

```r
s1=sample(nrow(d_train),0.7*nrow(d_train))
d_train_a=d_train[s1,]
d_train_b=d_train[-s1,]

set.seed(3)
gbm.layer1=gbm::gbm(y~.,
                    data=d_train_a,
                    distribution = "bernoulli",
                    n.trees = 2000,interaction.depth = 3,
                    n.minobsinnode = 5,
                    shrinkage = 0.05,
                    verbose=T,
                    n.cores=3)
k=gbm::gbm.perf(gbm.layer1,method="OOB")
k

d_train_b$gbm=gbm::predict.gbm(gbm.layer1,d_train_b,n.trees = k,type="response")
d_test$gbm=gbm::predict.gbm(gbm.layer1,d_test,n.trees = k,type="response")
log.layer2=step(glm(y~.,data=d_train_b,family="binomial"))
```

```
model_perf[model_perf$model=="log_pred",]
   model precision   recall        f1       acc
log_pred 0.4939759 0.600183 0.5419248 0.8773637
```

Improved performance

```
> precision;recall;f1;acc
[1] 0.5289855
[1] 0.7346752
[1] 0.61509
[1] 0.8888643
```

Stacking

- Stacking : Idea is to use CV and all your data. Here is one example with 5 folds
  - Use 4 folds to get out of sample prediction on each fold for first layer models
  - Build second layer model on the all 4 folds taken together .
  - Then build first layer model on all 4 folds taken together and make prediction on fifth fold
  - Get error for final [second layer model] on the fifth fold
  - repeat this as 5 fold CV for second layer models

Stacking

```
## stacking
d_train_b$gbm=NULL
gbm.a=gbm::gbm(y~.,
               data=d_train_a,
               distribution = "bernoulli",
               n.trees = 2000,interaction.depth = 3,
               n.minobsinnode = 5,
               shrinkage = 0.05,
               verbose=T,
               n.cores=3)
ka=gbm::gbm.perf(gbm.a,method="OOB")
gbm.b=gbm::gbm(y~.,
               data=d_train_b,
               distribution = "bernoulli",
               n.trees = 2000,interaction.depth = 3,
               n.minobsinnode = 5,
               shrinkage = 0.05,
               verbose=T,
               n.cores=3)
kb=gbm::gbm.perf(gbm.b,method="OOB")
```

Stacking counted..

```
d_train_a$gbm=gbm::predict.gbm(gbm.b,d_train_a,n.trees = kb,type="response")
d_train_b$gbm=gbm::predict.gbm(gbm.a,d_train_b,n.trees = ka,type="response")

d_train=rbind(d_train_a,d_train_b)
log.layer2=step(glm(y~.,data=d_train,family="binomial"))

gbm.full=gbm.b=gbm::gbm(y~.-gbm,
                        data=d_train,
                        distribution = "bernoulli",
                        n.trees = 2000,interaction.depth = 3,
                        n.minobsinnode = 5,
                        shrinkage = 0.05,
                        verbose=T,
                        n.cores=3)
kfull=kb=gbm::gbm.perf(gbm.full,method="OOB")
d_test$gbm=gbm::predict.gbm(gbm.full,d_test,n.trees = kfull,type="response")

log_pred=as.numeric(predict(log.layer2,d_test,type="response")>0.2)
```

```
> precision;recall;f1;acc
[1] 0.5289855
[1] 0.7346752
[1] 0.61509
[1] 0.8888643
```

```
> precision_stack;recall_stack;f1_st
[1] 0.5602679
[1] 0.6889296
[1] 0.61509
[1] 0.8970474
```

More Ideas to explore

ideas…

- Build a classification probability model on bins of continuos response
- Build a regression model for binary class
- Use patterns from unsupervised learning models as features

Summarising it All

Caution,Gains & Pitfalls

- Use Algos which are different from each other e.g. SVM, Dtrees , KNN, GBM , RF ,NN, ET etc
- Just average CV errors are not enough , also look at variance
- Consider the stacking process to be feature engineering without knowing explicit transformations for variable
- Be careful about CV implementation in stacking . Info leak doesn't generalise well at all.

Thank you!

Questions?