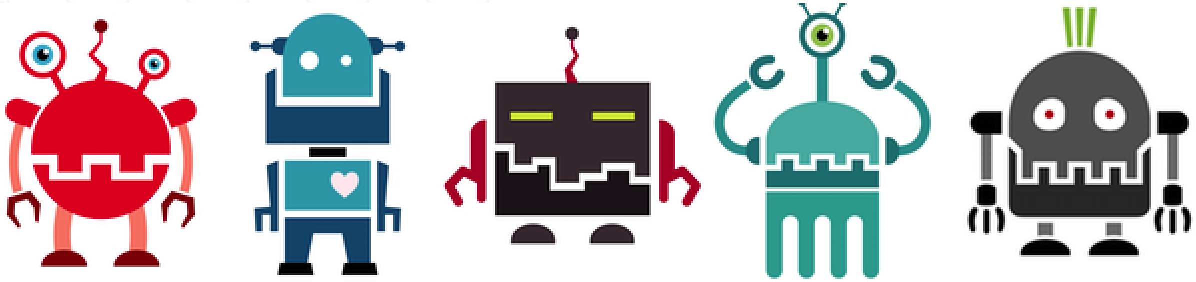
FACEBOOK: BOT OR NOT CHALLENGE

ALEKYA PYREDDY (903211489)
HIMADEEP REDDY REDDIVARI (903208781)
KIRTHANA HAMPAPUR (903198301)



Overview of the Presentation

- Problem Statement and Data Description
- Feature Engineering and Pre-processing
- Initial Methods: SVM, Neural Nets, Random Forest, Logistic Regression, AdaBoost
- Model Optimization and Comparison
- Handling Class Imbalance
- Ensemble methods - Stacking
- Two-Stage AdaBoost
- Conclusion and Future Scope

Problem Statement and Data

- Predict if an online bid is made by a machine or a human – data from Kaggle
- No meaningful data, mostly alphanumeric with transformed features

Bidder Information

- Bidder id – 91a3c57b13234af24875c56fb7e2b2f4rb56a
- Payment account - a3d2de7675556553a5f08e4c88d2c228754av
- Address - a3d2de7675556553a5f08e4c88d2c228vt0u4
- Outcome – 0 for human, 1 for bot

Bid Information – 7.6 million obsvervations

- Bidder id – alphanumeric data
- Auction id – alphanumeric data
- Merchandise – 10 levels
- Device – phone model
- Time - time that the bid is made (transformed)
- Country - country that the IP belongs
- IP Address - IP address of a bidder (transformed)
- URL, which the bidder was referred from - url where the bidder was referred from (transformed)

Feature Extraction – which features matter?

- Create features in order to compete – have to be creative
- Obvious features
  - Number of auctions placed by a bidder – more frequently place for a bot
  - Number of devices and IP addresses used – will be high for a bot
  - Number of distinct countries from which bids are placed – will be high for a bot
- Other features
  - Time difference between consecutive bids – low for bot
  - Country crime statistics – put in 5 tiers extracted from a database with SQL

```
> biddersdata[which(biddersdata$bidder_id == "17a321c4a0d925ca80507effa52330ac5n5r7")][1:30,]
     bid_id                             bidder_id auction merchandise    device             time country              ip             url
 1: 2351424 17a321c4a0d925ca80507effa52330ac5n5r7   qlnzb     jewelry  phone111 9631917789473684      rs    60.82.178.42 8alyseby9wbk3zr
 2: 2354077 17a321c4a0d925ca80507effa52330ac5n5r7   dvllu     jewelry  phone229 9631929105263157      ru 113.174.116.199 vasstdc27m7nks3
 3: 2356802 17a321c4a0d925ca80507effa52330ac5n5r7   mtg9u     jewelry   phone22 9631939684210526      nl  26.251.100.185 5roro09tppf5ixp
 4: 2358573 17a321c4a0d925ca80507effa52330ac5n5r7   strbn     jewelry  phone373 9631942526315789      cz  71.132.199.137 7pwsx2e55ckbccw
 5: 2359449 17a321c4a0d925ca80507effa52330ac5n5r7   hi7or     jewelry   phone59 9631943947368421      za   101.172.68.99 sfpt2p7p8zj436v
 6: 2367001 17a321c4a0d925ca80507effa52330ac5n5r7   tsbtt     jewelry  phone598 9631957421052631      de   96.110.81.233 vasstdc27m7nks3
 7: 2368100 17a321c4a0d925ca80507effa52330ac5n5r7   lwgcc     jewelry    phone6 9631959315789473      uk   6.192.198.142 5roro09tppf5ixp
 8: 2371667 17a321c4a0d925ca80507effa52330ac5n5r7   strbn     jewelry  phone300 9631965473684210      ph    111.3.138.93 h26spjh3n5pfxyu
 9: 2373285 17a321c4a0d925ca80507effa52330ac5n5r7   lwgcc     jewelry  phone300 9631968315789473      ph    111.3.138.93 h26spjh3n5pfxyu
10: 2375435 17a321c4a0d925ca80507effa52330ac5n5r7   uadbw     jewelry   phone17 9631972315789473      ph    129.17.66.57 zbvzsu3dnk70nk6
11: 2385466 17a321c4a0d925ca80507effa52330ac5n5r7   9ul86     jewelry  phone469 9631991315789473      uk   6.192.198.142 h26spjh3n5pfxyu
12: 2390878 17a321c4a0d925ca80507effa52330ac5n5r7   strbn     jewelry  phone129 9632001894736842      cz 159.253.145.159 95r1bz9d5ty7cle
13: 2397251 17a321c4a0d925ca80507effa52330ac5n5r7   lx0hm     jewelry   phone33 9632015947368421      ph  15.220.160.145 vasstdc27m7nks3
14: 2397823 17a321c4a0d925ca80507effa52330ac5n5r7   bf7f2     jewelry phone1586 9632017210526315      us   157.12.54.124 g5r5308duswuqad
15: 2406261 17a321c4a0d925ca80507effa52330ac5n5r7   9ul86     jewelry  phone212 9632036263157894      uk  117.22.153.104 77piptyiezdptc9
16: 2410890 17a321c4a0d925ca80507effa52330ac5n5r7   c1my1     jewelry  phone189 9632047526315789      my   11.139.116.12 vasstdc27m7nks3
17: 2412380 17a321c4a0d925ca80507effa52330ac5n5r7   fukqw     jewelry phone1586 9632051421052631      us    133.183.43.4 5roro09tppf5ixp
18: 2417033 17a321c4a0d925ca80507effa52330ac5n5r7   cz87a     jewelry phone1861 9632062842105263      ru 236.201.135.150 9bur171b8encqbo
19: 2417702 17a321c4a0d925ca80507effa52330ac5n5r7   9ul86     jewelry   phone76 9632064526315789      ua 179.140.215.143 vasstdc27m7nks3
20: 2423098 17a321c4a0d925ca80507effa52330ac5n5r7   uadbw     jewelry  phone212 9632078210526315      uk  117.22.153.104 9168a3122fv7v3s
21: 2423117 17a321c4a0d925ca80507effa52330ac5n5r7   uadbw     jewelry  phone212 9632078263157894      uk   6.192.198.142 9168a3122fv7v3s
22: 2423532 17a321c4a0d925ca80507effa52330ac5n5r7   u9gsl     jewelry   phone53 9632079421052631      th     12.83.18.21 mfbwizdplz3z98j
23: 2425235 17a321c4a0d925ca80507effa52330ac5n5r7   ip071     jewelry  phone252 9632083736842105      bn  167.65.234.108 vasstdc27m7nks3
24: 2431476 17a321c4a0d925ca80507effa52330ac5n5r7   dvllu     jewelry   phone45 9632099157894736      ru   63.225.63.156 vasstdc27m7nks3
25: 2432624 17a321c4a0d925ca80507effa52330ac5n5r7   3klk9     jewelry  phone252 9632102000000000      bn  167.65.234.108 9168a3122fv7v3s
26: 2433780 17a321c4a0d925ca80507effa52330ac5n5r7   lx0hm     jewelry   phone21 9632104842105263      by    81.54.92.252 vasstdc27m7nks3
27: 2433891 17a321c4a0d925ca80507effa52330ac5n5r7   3klk9     jewelry  phone252 9632105157894736      bn  167.65.234.108 9168a3122fv7v3s
28: 2434784 17a321c4a0d925ca80507effa52330ac5n5r7   9ul86     jewelry  phone258 9632107473684210      ph  233.172.46.204 vasstdc27m7nks3
29: 2436725 17a321c4a0d925ca80507effa52330ac5n5r7   9ul86     jewelry    phone5 9632112526315789      ph    41.81.80.193 vasstdc27m7nks3
30: 2437534 17a321c4a0d925ca80507effa52330ac5n5r7   nxfi3     jewelry phone1382 9632114842105263      tr 203.210.192.208 vasstdc27m7nks3
```

Feature Extraction – why features matter?

- Histograms

Feature Extraction – why features matter?

- Histograms

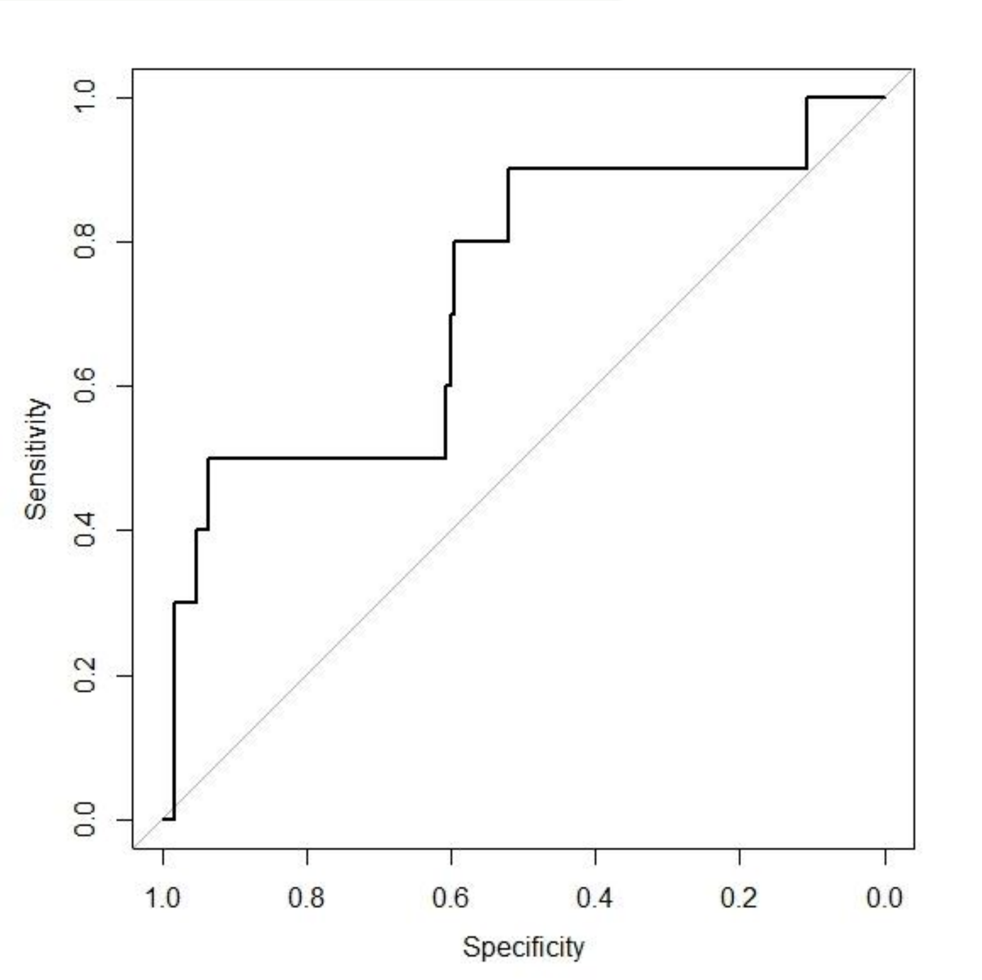
CREATING THE NEXT®

Feature Extraction – final features used

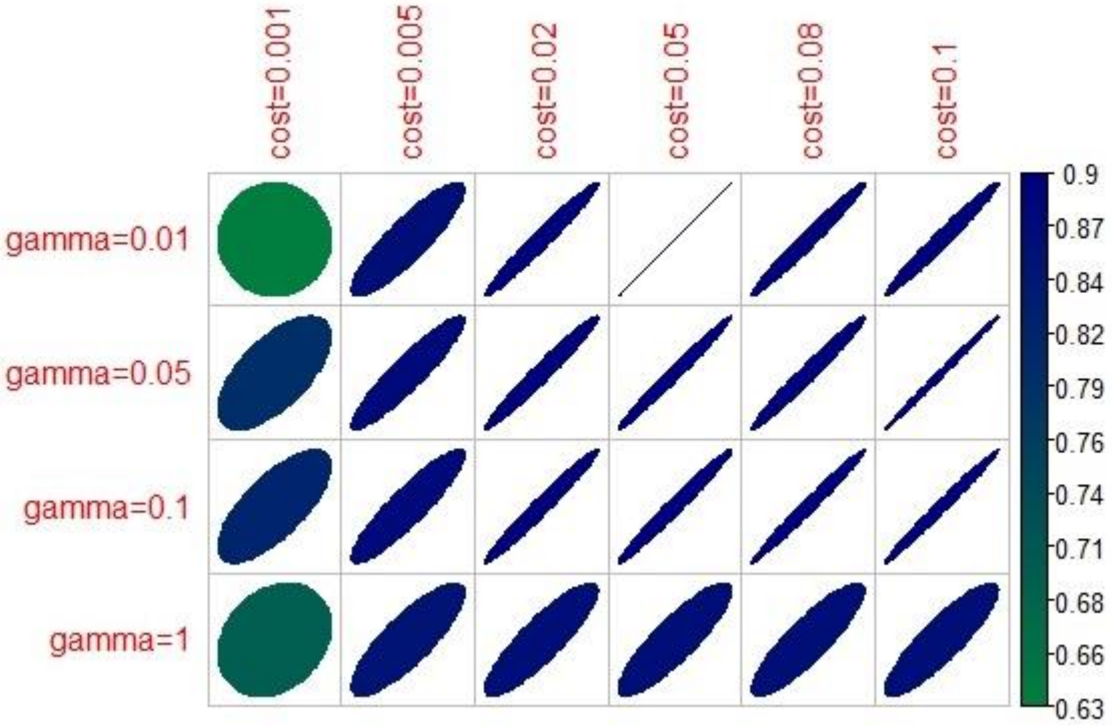
- Created database schema in SQLite and used RSQLite to get relevant features and statistics
- Bidder ID removed
- Mean, median statistics used, numeric features for merchandise

```
> head(myPredictors)
     bid_count country_count   ip_mean ip_median link_mean link_median device_count merchandise    tmean      tsd
[1,] 0.6931472             1 0.6931472 0.6931472 0.6931472   0.6931472            1           6 32.65626 32.30835
[2,] 0.6931472             1 0.6931472 0.6931472 0.6931472   0.6931472            1           8 32.65626 32.30835
[3,] 4.9558271            16 0.9842019 0.6931472 1.0140549   0.6931472           67           1 27.03079 29.07804
[4,] 1.3862944             2 0.6931472 0.6931472 0.6931472   0.6931472            3           9 30.94463 31.28923
[5,] 6.2989492            72 3.2046576 0.6931472 0.7503056   0.6931472          165           6 23.94505 24.69975
[6,] 3.1780538            10 0.7444405 0.6931472 0.7444405   0.6931472           16           8 27.05518 27.24176
      tmedian     tmin     tfast  max_bids max_country_count crime_mean_p crime_median_t crime_max_p crime_max_t
[1,] 32.65626 32.65626 0.0000000 0.6931472                 1   0.05039277              2  0.05039277           2
[2,] 32.65626 32.65626 0.0000000 0.6931472                 1   0.05407473              2  0.05407473           2
[3,] 24.75143 20.08141 0.9214286 1.0665729                16   0.18141547              3  0.54118408           5
[4,] 30.94463 24.71322 0.5000000 0.6931472                 2   0.29682565              4  0.47601808           5
[5,] 23.36608 17.77883 0.6845018 3.4947234                72   0.14953025              3  0.80239521           5
[6,] 26.88719 18.47197 0.6363636 0.7932306                10   0.09710505              3  0.20400406           4
>
```

SVM

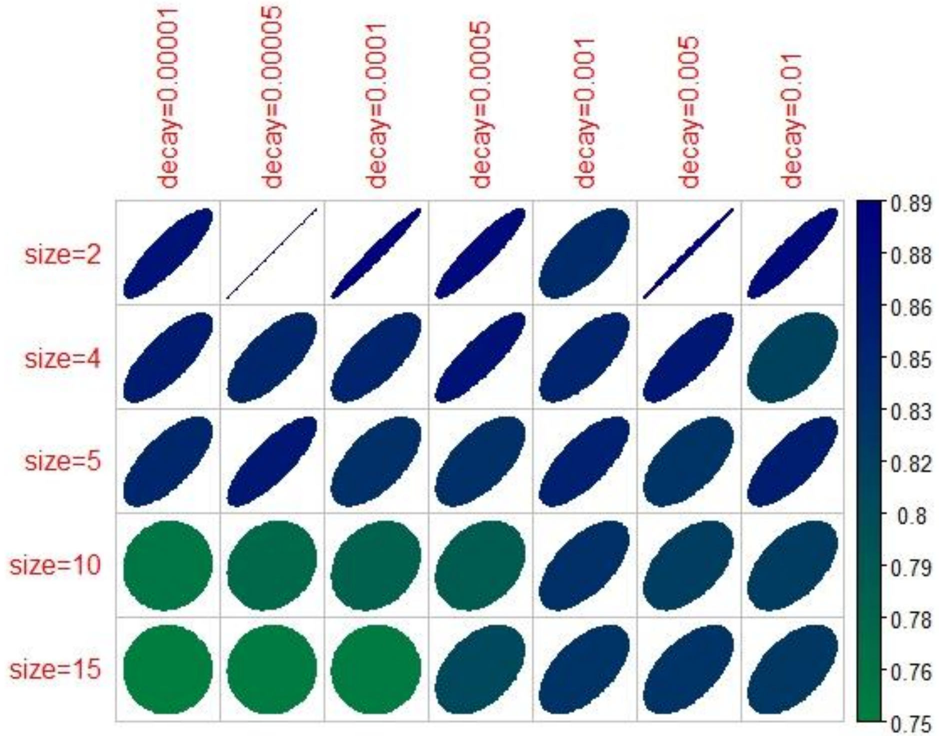


ROC curve for SVM

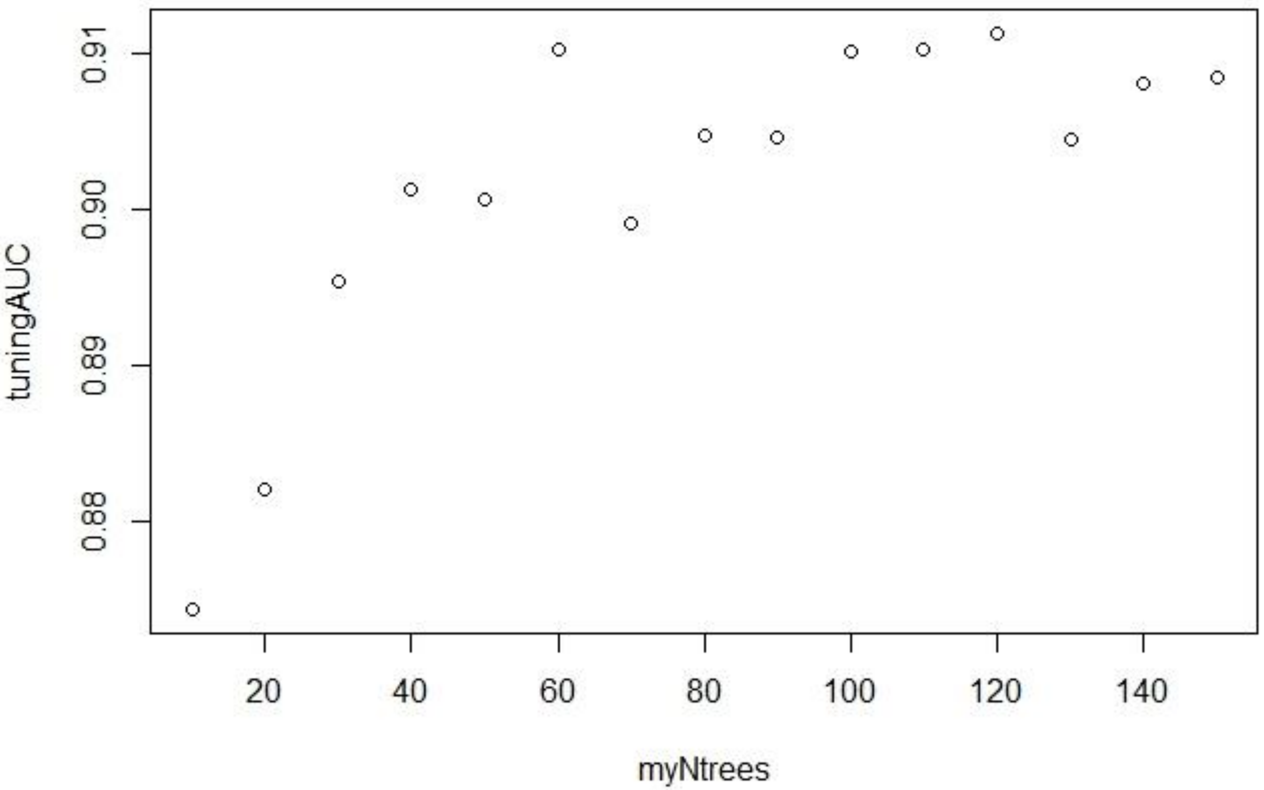


Parameter tuning for SVM

Neural Nets and Random Forest

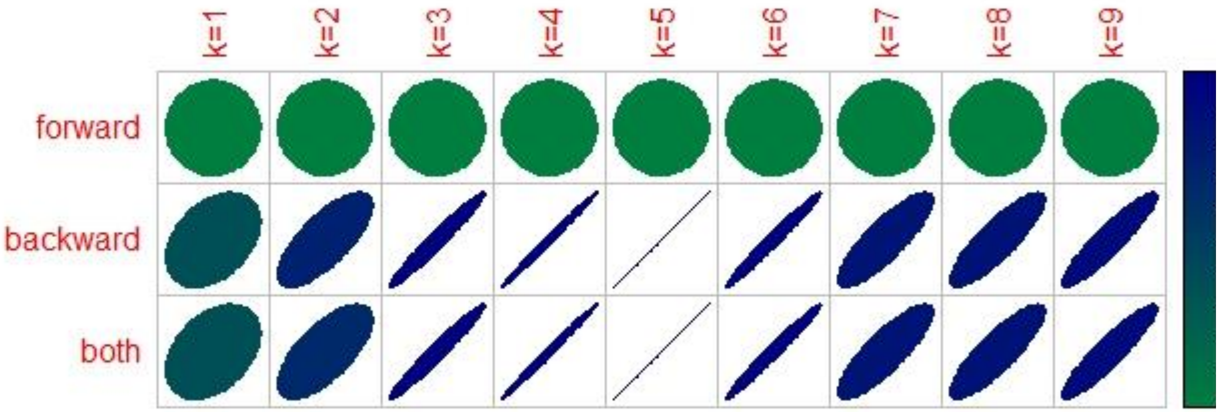


Parameter tuning for Neural Nets

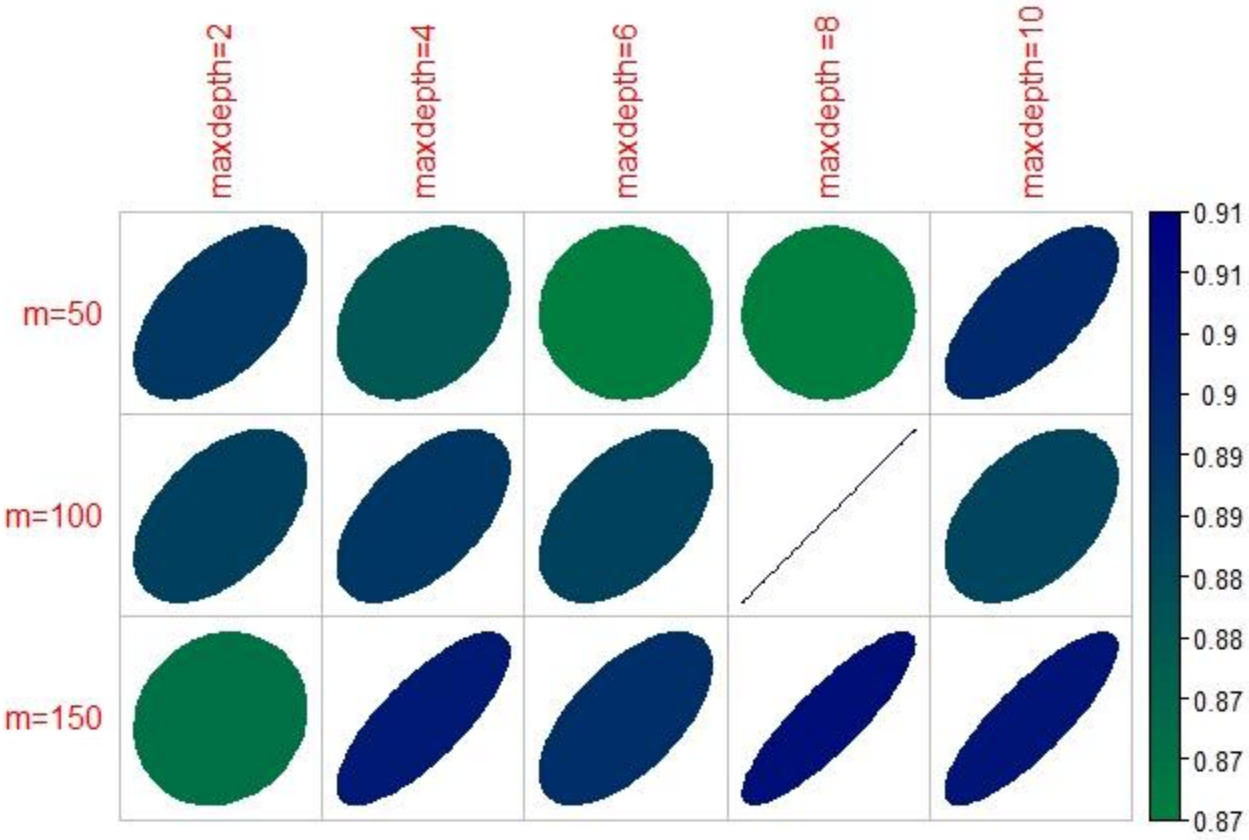


AUC vs number of trees in Random Forest

Logistic Regression and AdaBoost



Parameter tuning for Logistic Regression



Parameter tuning for Adaptive Boosting

Parameter Tuning and Model Comparison

Base Models	Parameter 1	Optimal	Parameter 2	Optimal	Training AUC	Submission AUC
SVM	Gamma	0.01	Cost	0.05	0.7881597	0.6019000
Neural Nets	Decay factor	0.00005	Units in hidden layers	2	0.8795238	0.7864244
Random Forests	Number of trees	120	--	--	0.9046143	0.8606039
Logistic Regression	K	5	Type of Regression	Both	0.7971601	0.6658645
Adaptive Boosting	Number of trees used	100	Maximum Depth of tree	8	0.9122576	0.8706247

Handling class imbalance

- Dataset is not balanced – leads to very bad predictions
- Data distribution has to be taken into consideration
- Standard learners are often biased towards the majority class
- SMOTE – Synthetic Minority Oversampling Technique
- SMOTE = under sampling from minority class + over sampling from majority class
- Minority sample
  - Find k-nearest minority neighbors
  - Randomly select j and generate new samples and join to minority sample
- Disadvantage – can over generalize, number of sample fixed, no flexibility

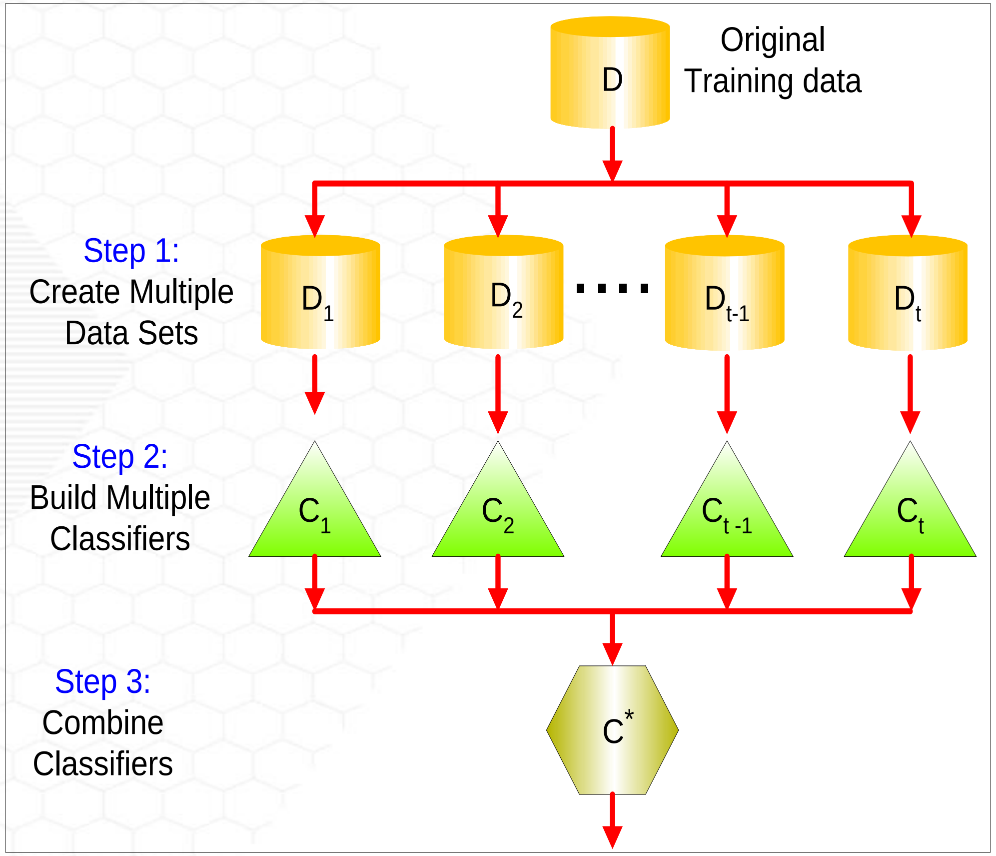
Results - SMOTE

- Initial proportion of outcomes – 94.8% zeros, and 5.2% ones
- After SMOTE – data is equally balance (50:50)
- Results not as expected – basic models still fail
- Justified? – yes
- The data may not be completely representative of the population
- No better than random guessing
- To avoid over-generalization we can cluster the minority data, or do adaptive sampling techniques

Stacking

- Generate group of base learners and combine for better accuracy
- Rationale – base algorithms might be making assumptions not valid for the dataset
- Useful when there is base models with different kinds of errors

Rule	Fusion function $f(\cdot)$
Sum	$y_i = \frac{1}{L}\sum_{j=1}^{L} d_{ji}$
Weighted sum	$y_i = \sum_j w_j d_{ji}, w_j \geq 0, \sum_j w_j = 1$
Median	$y_i = \text{median}_j d_{ji}$
Minimum	$y_i = \min_j d_{ji}$
Maximum	$y_i = \max_j d_{ji}$
Product	$y_i = \prod_j d_{ji}$



Stacking

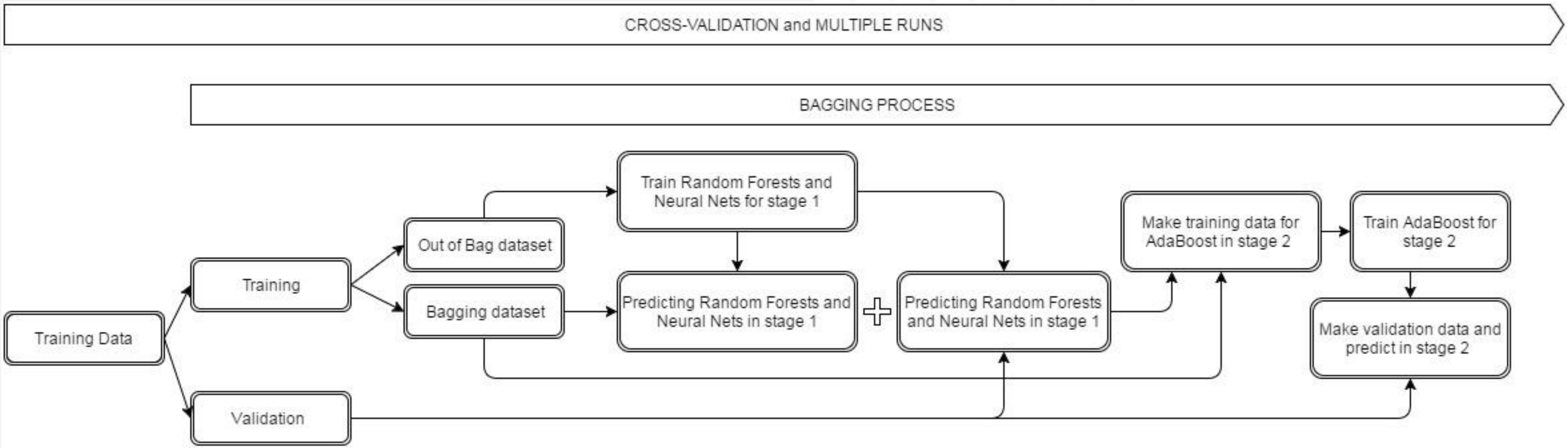
```
> load("\\\\prism.nas.gatech.edu\\hreddivari3\\vlab\\documents\\cdaproject\\facebook3\\allmodels.RData")
> D = 2*t(miuMatrix)%*%miuMatrix
> d = 2*t(miuMatrix)%*%yMatrix
> A = t(rbind(matrix(1,1,5),matrix(-1,1,5),diag(c(1,1,1,1,1))))
> b = c(1, -1, 0, 0, 0, 0, 0)
> library(quadprog)
> solution = solve.QP(D,d,A,b,factorized = FALSE)
> weight1
[1] 0.00000000 0.29739746 0.67857403 0.00309332 0.02093519
> miuMatrix
              [,1]         [,2]       [,3]       [,4]         [,5]
 [1,] 0.0077389382 7.959216e-05 0.00000000 0.07409190 0.0026699788
 [2,] 0.0198678285 1.154559e-03 0.00000000 0.17940736 0.0093000844
 [3,] 0.0392386043 2.746689e-03 0.00000000 0.28246078 0.0167904119
 [4,] 0.0230577887 6.252301e-04 0.00000000 0.13646921 0.0147351710
 [5,] 0.0179972077 2.649493e-04 0.00000000 0.14983006 0.0086197719
 [6,] 0.0276424145 1.155102e-03 0.00000000 0.07971587 0.0253797903
 [7,] 0.0619790387 1.029393e-03 0.00000000 0.25960431 0.0008105304
 [8,] 0.0513445679 3.091730e-02 0.00000000 0.25799776 0.0499925360
 [9,] 0.0310843096 1.154419e-03 0.00000000 0.20921669 0.0044335091
[10,] 0.3662477663 9.524079e-01 0.53333333 0.57783652 0.4726344466
[11,] 0.0401549457 1.390689e-01 0.00000000 0.21971272 0.0235723879
[12,] 0.0280439615 1.188399e-03 0.00000000 0.14790726 0.0066072308
[13,] 0.0324023428 3.090160e-02 0.00000000 0.24752348 0.0577206415
[14,] 0.1551070044 3.049611e-01 0.26666667 0.39793976 0.2244573957
[15,] 0.0777508259 3.229288e-01 0.06666667 0.17647941 0.1001942542
[16,] 0.0538042241 1.145005e-03 0.00000000 0.31138276 0.0010190740
[17,] 0.0426191954 8.567836e-02 0.06666667 0.25319676 0.0352887783
[18,] 0.0109266973 1.069331e-03 0.00000000 0.14908681 0.0223913019
[19,] 0.0403855823 8.169717e-04 0.00000000 0.17344853 0.0711860144
[20,] 0.0349037445 6.865696e-03 0.00000000 0.31102402 0.0332886340
[21,] 0.0291308552 9.557370e-05 0.00000000 0.11360993 0.0091701246
[22,] 0.0076590872 1.203512e-01 0.06666667 0.13170062 0.0169448766
[23,] 0.0352348818 1.329780e-03 0.00000000 0.11886045 0.0091823874
[24,] 0.0462035413 2.982112e-02 0.06666667 0.26874687 0.0599478157
[25,] 0.0545803563 2.377410e-02 0.06666667 0.35717625 0.0300004369
[26,] 0.0342357231 1.567589e-02 0.06666667 0.29161418 0.0370696488
[27,] 0.0281557642 1.155522e-03 0.00000000 0.20706034 0.0046353770
[28,] 0.0325438687 1.154456e-03 0.00000000 0.21510924 0.0074373880
[29,] 0.0500036185 1.491171e-02 0.33333333 0.25028700 0.1109262997
[30,] 0.0615018061 9.372565e-04 0.00000000 0.34086063 0.0007996718
```

- Formulate objective function as minimization of sum squared errors from base models on validation sets
- Constrained weights obtained
- Prediction using weighted sum combination rule
- Stacking model improves submission scores from 0.87 to 0.89
- Train a single layer logistic regression model to use as a combiner

2 stage AdaBoost

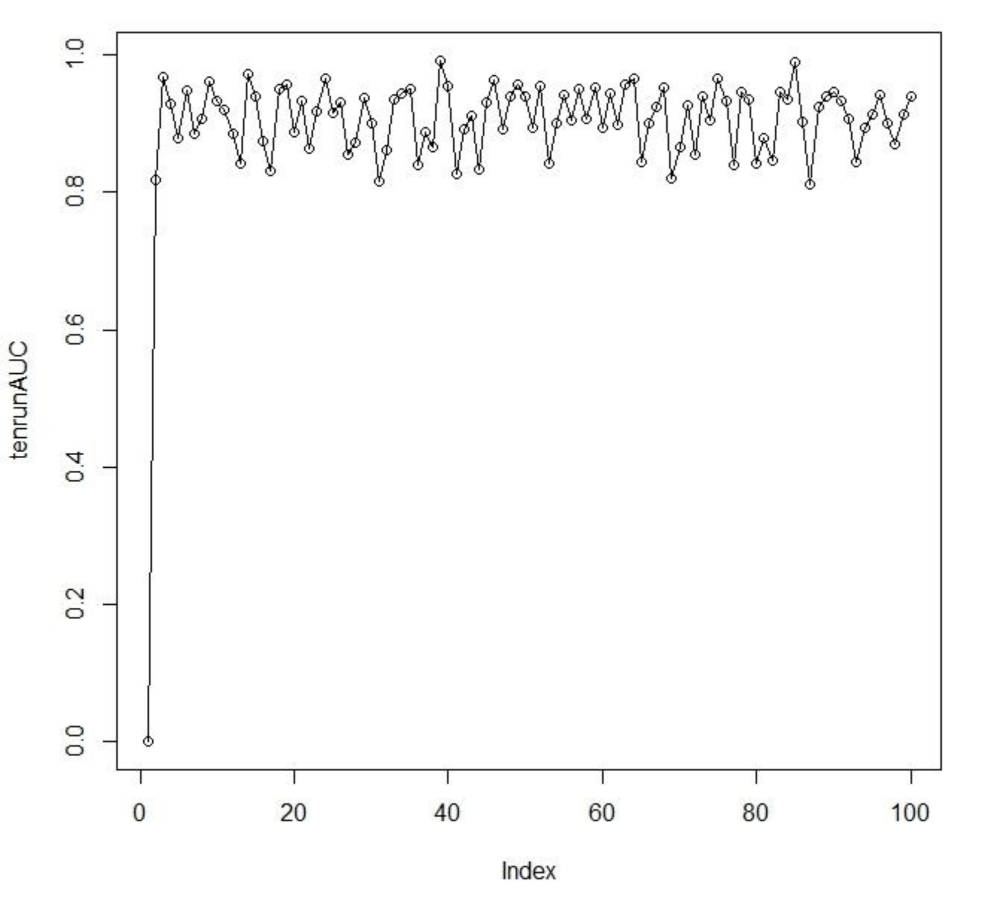
- Can we perform better than stacking?
- 2 stage AdaBoost with bagging
- Stage 1 – RF and Neural nets on out of bag data (OOB)
- Stage 2 – AdaBoost trained on bag data
- Use predictions from stage 1 to train AdaBoost stage 2
- 3:1 split on Bag and OO-Bag data
- 10 fold cross-validation done for 100 runs to check stability

2 stage Ada-Boost – Structure



- Predictions from random forests and neural nets model used as features to train AdaBoost model in stage 2
- Features added is $\sqrt{probability\ from\ random\ forest\ \times probability\ from\ neural\ nets}$
- If both probabilities are almost same we add similar feature, and if not product takes care of it

2 stage Ada-Boost – Results




```
+
+       pred_final = pred_final + pred
+       #pred_final = pred_final + rank(pred, ties.method = "random")
+
+     } # end of bagging
+     #pred_final = seq(from=0, to=1, length.out=nrow(valid_x))[rank(pred_final, ties.method = "random")]
+     pred_final = pred_final / z
+     auc_scores[cv] = evaluation(pred_final, valid_x[,ncol(valid_x)])
+     cat(auc_scores[cv], " ")
+   } # end of CV
+   cat("\n")
+   print(auc_scores)
+   cat("mean: ", mean(auc_scores), "sd: ", sd(auc_scores))
+   total_scores[r] = mean(auc_scores)
+ }
Scores: CV:  1
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  0.8185185  CV:  2
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  0.9666344  CV:  3
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  0.9284333  CV:  4
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  0.8781431  CV:  5
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  0.9473404  CV:  6
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  0.8856383  CV:  7
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  0.9066489  CV:  8
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  0.8787234  CV:  9
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  0.962234  CV:  10
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  0.9319149
 [1] 0.8185185 0.9666344 0.9284333 0.8781431 0.9473404 0.8856383 0.9066489 0.8787234 0.9622340 0.9319149
mean:  0.9104229 sd:  0.04608578Scores: CV:  1
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  0.9193122  CV:  2
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  0.8853965  CV:  3
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  0.8423598  CV:  4
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  0.9719536  CV:  5
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  0.9398936  CV:  6
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  0.875266  CV:  7
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  0.831117  CV:  8
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  0.95  CV:  9
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  0.9574468  CV:  10
1  2  3  4  5  6  7  8  9
```

Conclusions and Improvements