Davidson Shahar – 203256128 – [davidson.shahar@gmail.com](mailto:davidson.shahar@gmail.com)

Sheffer Roee – XXX - [bluewaffles@gmail.com](mailto:bluewaffles@gmail.com)

**Project – introduction to machine learning**

How to run the final main script:

XXX

Results summary:

Average success rate (over snps), 6-fold cross validation.

Each algorithm is descried in the future algorithm section.

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Success Rate (%) | .m file | Best parameters |
| Boosted algorithm |  | Go.m |  |
| Alternatives: | | | |
| K-Nearest Neighbors |  | go\_nearest\_neighbor.m |  |
| SVM |  | go\_svm.m |  |
| SVM2 (pre-processing) |  |  |  |
| Decision Tree |  |  |  |
| Adaboost |  |  |  |
| Histograms descriptor + svm |  |  |  |
| "Entangled snp" |  | go\_entangled\_snp.m |  |

Few notes:

* Evaluating the algorithms:
  + Success of single snp prediction over all test samples: 0-1 loss function, comparing prediction with ground-truth.
  + Success rate of algorithm: average of success rate of all snps.
  + Using 6-fold cross validation.
* part 1: implement and tune several different algorithm.
  + Using the same algorithm and same parameters over all snps, training 300 models (for each snp).
  + Check the success rate of the algorithm.
  + Search for best parameters.
  + Examine the success rare histogram of the different snps.
* part 2: Boosting.
  + For each snp, we can train models using many algorithms from part 1.
  + Examining the success rate of different algorithms, we noticed that for some snps algorithm A is giving better results (6-fold validation), and for some it would be B.
  + The final model is taking the best model and best parameters for each snp.
  + We didn’t have time to implement the next step – better boosting: for example, for each snp train many models, and performing adaboost on them. Our the code framework absolutely enable it, create unify API for each model class.
* Analysis of data:
  + At the beginning we check for correlation (0-1 loss function) between each missing snp, and all other (~165k). For each missing snp, we sorted the indexes of the best correlated snps.
  + It gave us a strong indication that missing snp is correlated mostly with it's near environment. This was very important for the running time of the algorithms - ignoring far snps. It also gave us an estimation of what is "far".
* Large group of snp's were hard to predict for every algorithm we tried. ( < 60%).
  + At the end we tried developing algorithm specifically for this subset of snps.
  + Without significant improving.
  + We wonder if it is due to not successful feature selection, or due to randomness in those snps values.

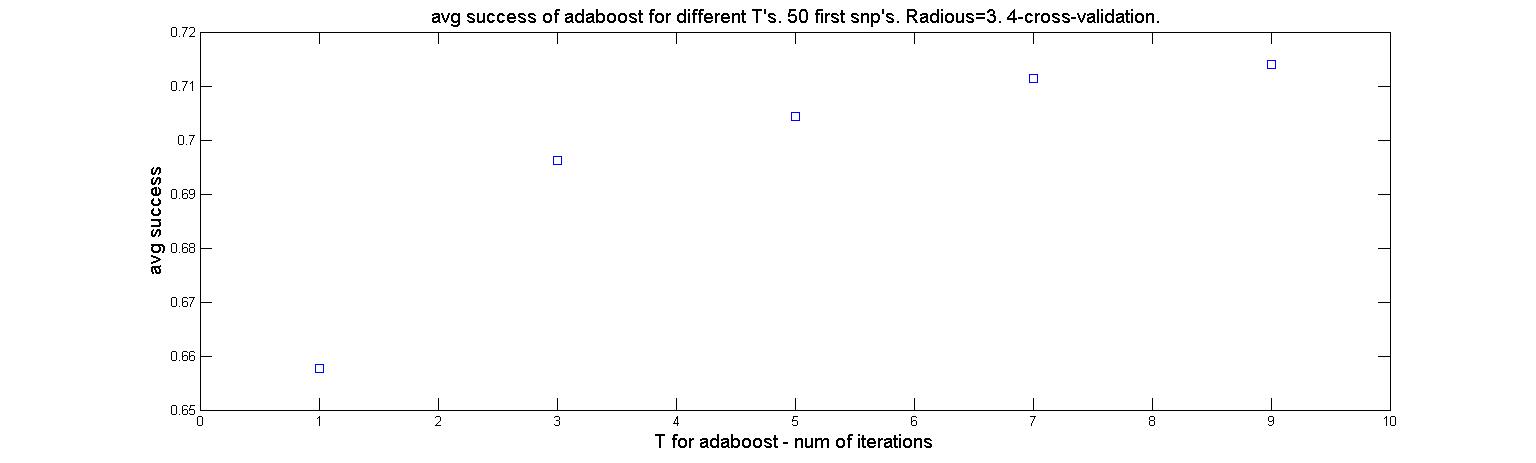
Algorithms Description:

* K-Nearest Neighbors:
  + Predict snp i of person j: find the k nearest neighbors (L2 norm) from training set. Prediction according to their label.
  + Vector representation of missing snp: vector of 100 snp's before it, and 100 after it.
  + We found that snps has local affection on other snps (correlations). Thus, we want the closest snp's to affect the most on the missing one. The algorithm multiply the vector with 1-d Gaussian, weighting the near snps.
  + Parameters:
    - K – for KNN
    - Sigma – for the weighting Gaussian. Bigger sigma -> bigger effect of wide windows around the missing snp
* SVM:
  + Again, representing a missing snp, with vector of it's R nearest snp's from each side. i.e: 100. We noticed that near snp affect more than far, and therefore the Radius R.
  + For each missing snp, a different svm model was trained.
  + Using libsvm.
  + Parameters:
    - R – radius of window around the missing snp
    - Svm options – passed to libsvm
* SVM2:
  + The same as previous svm, but with smarter features selection.
  + During the training stage, For each missing snp, find it's correlations (including permutations of 0 1 2) with near snps. Take as features the best X correlated one.
  + Continue with libsvm.
  + Parameters:
    - R , svm options – like previous SVM algorithm
    - X – number of top correlated near snp's to take.
* Decision Tree:
  + Using matlab implementation: ClassifiacationTree
  + Create for each snp a decision tree, pruned to level 2 (matlab api).
  + Consider only snps that are within R window around the missing one.
  + Parameters:
    - R – radius
* Adaboost:
  + Our implementation.
  + Weak classifiers are: look at the snp in the index i, check if it's value is j.
  + For each snp: trained 3 binary classifier: 0 or else, 1 or else, 2 of else. On test sample, run the three of them, taking the label with biggest coefficient result (before taking only the sign).
  + Parameters:
    - R – radius
    - T – number of iterations for adaboost
* Histograms descriptor + SVM:
  + tried to find different features vectors for the SVM.
  + Histograms descriptor: taking a window of snps around the missing snp. Dividing it to buckets, and create a vector from the histograms of each bucket.
  + Parameters:
    - Width – of the window.
    - Slide interval – size of each bucket inside the window.
* Entangled SNP:
  + This algorithm searches for the most "entangled" SNP to each missing SNP
  + The most entangled SNP is found to be the SNP (or any of the 27 substitutions of the SNP, [0 1 2] -> [? ? ?]), within the radius before and after the missing SNP whose percentage of different values is lowest.
  + Once found, classifying is done by simply choosing the value of the entangled SNP after substitution.
  + Paramter:
    - Radius

Submitted Files:

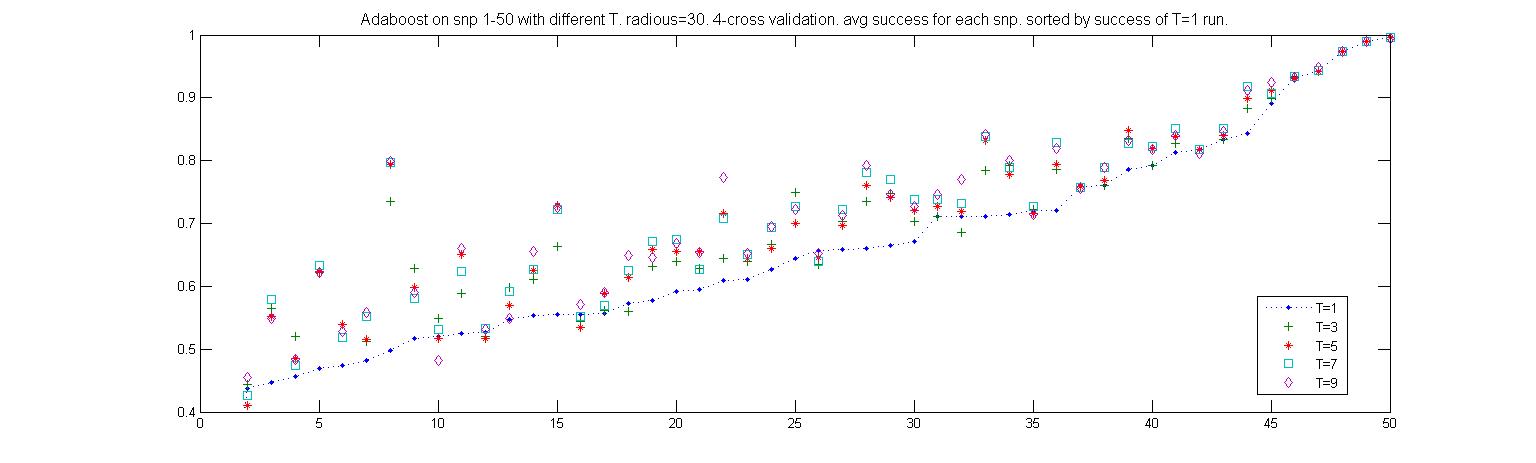
* Readme.pdf – this file. The report.
* Bin folder – contains some binaries, such as libsvm
* Src folder:
  + Algorithms folder: implementations of the algorithm mentioned above.
  + Go.m – main script. Boosting of all implemented algorithms.
  + Go\_generic.m – this function receive some learning algorithm instance. It loads the data, trains a model, uses it to predict the test sample's labels, and save them to result file – ytest.mat
  + Go\_\*.m – alternative learning algorithms. Using one of the mentioned algorithm above, and pass I to go\_generic()
  + Setup.m – handle loading data and adding pathes to matlab

Some graphs:



We choose to provide some graphs for one algorithm – adaboost:

* We examine how number of iterations (adaboost) affect the resulted success rate.



* The first graph shows how the average success rate increase as function of T. It also shows that the improving decreases at T~=10
* The second graph shows that some snp are boosted well when using more iterations of adaboost. But it is interesting to see, that some are not improved, and even get worse.