# Introduction to Machine Learning – Final Project

## Submitted by:

Shahar Davidson - shahard1 - 203256128 - [davidson.shahar@gmail.com](mailto:davidson.shahar@gmail.com)

Roee Sheffer - roysheffer - 302232269 - [roee.sheffer@gmail.com](mailto:roee.sheffer@gmail.com)

## Usage:

Before any scripts can be run, the train.txt, test.txt and dataforproject.mat need to be placed in the data folder

… TODO

## Notes:

### Algorithm Framework:

In order to enable easy implementation and comparing of different algorithms, we implemented each algorithm as a matlab struct with the following fields: train, classify, description, and params.

params – algorithm parameters and constants in a struct.

description – a description of the algorithm used for indicative prints when testing with multiple algorithms.

train – A function pointer to the algorithms train function. This function receives the training data and params as input and returns a model struct that differs per algorithm.

classify – A function pointer to the algorithms classify function. This function receives the test data and the previously trained model and returns the labels given by the model.

Implementing a new algorithm is as easy as copying the template algorithm and filling in the above fields and testing it is done by giving the algorithm struct to the cross-validation function as an argument. Also, in the next sections we shall describe our multi-algorithm approach for each SNP and this is made easy by the above interface.

### Evaluating the Algorithms:

Our algorithms were evaluated by comparing the 6-fold cross validation accuracy with a 0-1 loss function.

All of our algorithms used only the local data (extracted\_train and extracted\_test) and trained classifiers per SNP, allowing us to calculate accuracy per SNP. The resulting accuracy of the algorithm is the average accuracy upon all SNPs.

While testing, we noticed that different algorithms succeed better on different SNPs. The   
“per SNP” nature of all of our algorithms enabled us to choose the best algorithm / parameters for each SNP and estimate the missing test data using the optimal algorithm and parameters for each SNP.

Another method we thought of using but didn’t implement, is classifying using the majority of the 3 (or any k) best algorithms for each SNP.

### Data Analysis:

Before implementing any of the algorithms, we calculated the correlation of all missing SNPs in the training set with all other SNPs in order to empirically discern if SNPs are locally dependent and if they are, what the scale of an SNPs local environment is. We found that SNPs that are strongly correlated are usually very close (distance of less than 10 SNPs) and that some of the SNPs are not correlated strongly to any other.

Based on this, most of our algorithms use only a small amount of SNPs surrounding the SNP we are trying to learn. This achieves better accuracy as well as better running times.

Once we arrived at several good algorithms, we tried to find an algorithm that successfully predicts the SNPs that we were bad at predicting (less than 60% accuracy). Although trying many different methods, we failed to achieve good results predicting these SNPs and we attribute that to incorrect feature selection, independent data, and lack of understanding the physics of the problem.

## Algorithms:

### KNN:

We used the matlab KNN implantation and support different values of K as well as different distance metrics. Initially we tried using only the D values surrounding a certain SNP but achieved better values by multiplying the data by a one dimensional Gaussian vector whose mean is at the missing SNP and variance is given as a parameter to the algorithm. (The smaller the variance, the less weight is given to the SNPs farther away from the missing one).

Parameters: K, sigma

### SVM:

Classification is done using libsvm (multi-class). The training data is the R (a parameter to the algorithm) SNPs closest to the missing SNP.

Parameters: dim\_radius (R) and the libsvm options (kernel type, C and so forth)

### SVM2:

Same as before, but the feature vector is composed of the K SNPs most correlated to the missing SNP within a given dim\_radius.

Parameters: dim\_radius, top\_features (K), svm\_options

### Entangled SNP:

In this algorithm we look for the SNP most “entangled” to the SNP we are trying to predict. The most entangled SNP is found to be the SNP (or any of the 27 substitutions of the SNP, [0 1 2] -> [? ? ?]) most correlated with the missing SNP. Once found, classifying is done by simply choosing the value of the entangled SNP after substitution. As before, we look for the most correlated SNP only with a certain distance from the missing SNP.

Parameters: dim\_radius

### Decision Trees:

We used the matlab implementation: ClassificationTree, with 2 levels pruned. Like before, we consider only dim\_radius SNPs surround the missing SNP.

Parameters: dim\_radius

### Histogram Descriptor + SVM:

This algorithm is a result of us looking for more interesting feature vectors for SVM classification and manually looking at the data of the SNPs we failed to predict with high accuracy.

The histogram feature vector is composed of the concatenation of histograms of different sequences of SNPs within the extracted data. The sequences of data upon which the histogram is calculated is defined by the arguments width (length of the sequence to calculate the histogram on), and slide\_interval (the interval between sequences for which we calculate the histogram). We start by calculating the histogram of the sequence in the middle of the extracted data, and then we slide, slide\_interval SNPs, in each direction and calculate the next 2 histograms until we reach the end of the data.

Libsvm is used in the same manner as the previous SVM based algorithms.

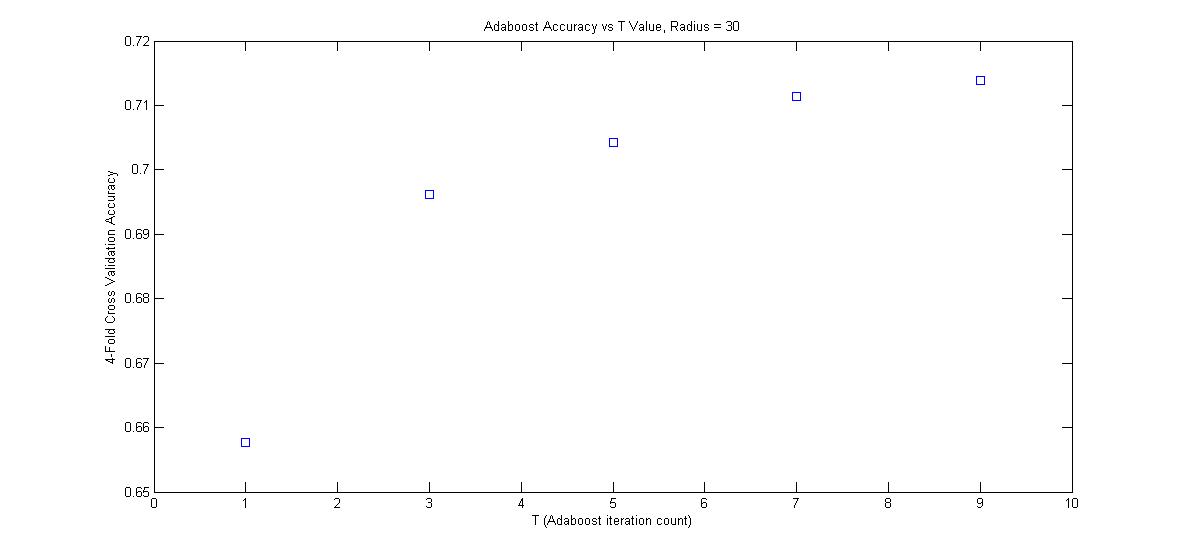
Parameters: width, slide\_interval, and svm\_options

### Adaboost:

Our implementation of adaboost. Multi-class classification is done by training 3 models for each value vs all the rest. The value chosen is the value with the highest confidence value. The weak classifiers used are decision stumps (for example: classify 1 if the value at index j is k).

Parameters: T (adaboost iteration count) and dim\_radius

In order to test our Adaboost, we created several graphs the show the accuracy level vs the number of boosting iterations. We can see that the average accuracy, and accuracy per SNP increase as we run more boosting iterations but the graph eventually flattens out.



## F:\Roee\Documents\Studies\Year 2\Semester A\Machine Learning\Exercises\final_project\parsnip\results\adaboost results.jpg

## Results:

As we specified before, the algorithms were originally ranked according to their 6-fold cross validation accuracy but in our final algorithm, we calculate the 6-fold cross validation accuracy for each SNP separately and choose the best algorithm for each missing SNP.

The success rates described below are for the same algorithm used across all SNPs while the success rate of the boosted algorithm is the average success rate upon all SNPs after choosing the best algorithm for each SNP. The algorithms tried in go.m are each of the algorithms with the parameters shown below and some other parameters that we found to be successful while testing.

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Success Rate (%) | .m file | Best parameters |
| Boosted algorithm | 79.81 | go.m |  |
| Alternatives: | | | |
| KNN | 75.79 | go\_nearest\_neighbor.m | K=3, sigma=3, distance=’euclidean’ |
| SVM | 79.02 | go\_svm.m | radial kernel with g=0.007 and c=100, radius 9 |
| SVM2 (pre-processing) | 74.42 | go\_svm2.m | radial kernel with g=0.01 and c=100, radius=30, top\_features=10 |
| Decision Tree | 75.98 | go\_tree.m | radius=8 |
| Adaboost | 74.45 | go\_adaboost.m | radius=40, T=10 |
| Histogram Descriptor | 69.54 | go\_histogram.m | quadratic kernel with c=1, width=4, slide\_interval=1 |
| Entangled SNP | 69.77 | go\_entangled\_snp.m | dim\_radius = 20 |

### 

### Submitted Files:

readme.pdf – This file. The project report.

bin\ - This folder contains binary data needed for our algorithms. Currently it contains the windows and linux builds of libsvm.

data\ - This folder contains the exercise data. This folder is empty upon submission but train.txt, test.txt and dataforproject.mat should be placed here before running the scripts.

results\ - The resulting ytest.mat files from the go\_< algorithm>.m functions are created in respective folders with the name of the algorithm. The result of go.m is saved as ytest.mat and chosen\_algorithms.mat in the root of this folder.

src\ -

* cross\_validate.m – cross validation of algorithm that implements algorithm interface
* setup.m – loads project data
* go\_generic.m – calculates ytest for algorithm that implements algorithm interface
* go.m – calculates ytest by finding the best algorithm per SNP, as described previously
* go\_<algorithm>.m – one for each algorithm. uses go\_generic with algorithm initialized with best parameters we found for the specific algorithm

src\algorithms\

* <algorithm>\_algorithm.m – implementation of all algorithms