Improvement of Pareto Optimal Classification Via Bagging and Boosting

Jiacheng Weng 20576707

1. Introduction

Classification is a type of problem that matches samples with correct labels. Specifically, a two-class or binary classification is a subset of classification problems in which samples only have positive and negative labels. In a standard classification problem, a classifier is optimized to generate the minimum prediction error rate. There exist many algorithms that can perform classification tasks. When a data set is small, Kernel SVM can achieve high prediction accuracy (Cortes, 1995). Extreme Gradient Boosting is a gradient boosted tree method that often achieves state-of-the-art performance in classification tasks (Chen, 2016). When a dataset is large and complex, large learning systems such as neural networks with feature extractions are often applied.

1.1 Trade-off in Binary Classification

In some fields such as medical applications, the prediction rates of the true positive and the true negative cases are important. The analysis of the trade-off between true positive and true negative cases is called *cost-sensitive learning* (Bach, 2005). In this case, the performance of a binary classifier h can be summarized into a pair $(P_{tp}(h), P_{tn}(h))$ that represents the prediction probability of true positive and true negative cases. Given a set of sample-label pairs $< x, y >$, $(P_{tp}(h), P_{tn}(h))$ can be represented as follows:

$$P_{\mathrm{tn}}(h) = \Pr(h(x) = -1 \mid y = -1)$$
$$P_{\mathrm{tp}}(h) = \Pr(h(x) = +1 \mid y = +1)$$

With the formulation of a standard classification problem, adjusting the trade-off between $P_{tp}(h)$ and $P_{tn}(h)$ is difficult to achieve. For complex algorithms such as neural networks, this problem becomes computationally intractable. There is a need to construct algorithms that can perform cost-sensitive learning efficiently.

1.2 Pareto Optimal Linear Classification

A Pareto optimal linear classification problem computes the optimal linear classifier based on the assumption of Gaussian class-conditional distribution (Kim, 2006). The Pareto optimality is achieved when the prediction probability of true positive cases or true negative cases cannot be further improved without reducing the other. By formulating the training process into a convex optimization problem, the Pareto optimal linear classifier can be obtained efficiently which makes the cost-sensitive learning feasible (Kim, 2006). However, there exist two problems with the current method. One is that the kernel implementation of the Pareto optimal classifier is computationally expensive when the data set is large. The other one is that the empirical optimal trade-off curve produced by the Pareto optimal linear classifiers is asymmetric when the data set is not well balanced.

1.3 Brief Overview

In this report, the bagging algorithm and a modified Adaboost algorithm are implemented to address these two problems stated above. The bagging algorithm trains small Pareto optimal classifiers on bootstrap sub-data set which effectively reduce the computational complexity of the kernel methods. The modified Adaboost algorithm adaptively modifies the sample weights during training which produces more symmetric optimal trade-off curves than the reference approach.

2. Literature Review

2.1 Linear Classifier and Perceptron

A linear classifier follows the form $\hat{y} = sign(a^T x + b)$, where a is the weight vector and b is the bias. The original perceptron algorithm from Rosenblatt allows iterative learning of the weights and bias of this linear classifier by using training data (Rosenblatt, 1958). However, the optimality of the converged classifier only achieves sub-optimal solutions as the decision boundary can be close to the convex hull of two classes.

2.2 Artificial Neural network

An artificial neural network is a general machine learning architecture that is widely used on complex tasks such as speech recognition natural language processing. It is constructed by stacking multiple layers of perceptron with non-linear activation in between (Rumelhart, 1985). With the power of backpropagation, such complex systems can optimize their internal weights and biases by learning from error signals (Rumelhart, 1986). However, Neural networks generally require a large amount of training data and computing power to achieve convergence. In addition, they are prone to overfitting in small data sets.

2.3 Support Vector Machine

Support Vector Machine (SVM) is also a linear classifier that maximizes the margin or distance between the decision boundary and the convex hull of two classes (Cortes et al., 1995). As a result, SVM produces more generalized classifiers with generally higher prediction accuracy on unseen data than perceptron. Kernel trick is also introduced in kernel SVM which significantly increases classification accuracy by mapping samples to kernel feature space with some assumption of the underlying data distribution (Cortes et al., 1995). However, kernel SVM runs slow on large data set as it scales quadratically with the number of data points. Some methods such as EnsembleSVM address this problem by breaking down a large kernel to ensembles of small kernels (Claesen, 2014).

2.4 Pareto Optimal Linear Classifiers

Cost-sensitive learning in binary classification, which considers the trade-off between false positive and negative rates, is often not considered in classifier design due to its computational complexity (Kim, 2006). The Pareto optimal linear classifier, on the other hand, manages to perform cost-sensitive learning efficiently by assuming Gaussian class-conditional distribution, yet also achieves comparable results as SVM (Kim, 2006). The Pareto optimality of a binary classifier is achieved when true positive or negative rates cannot be further increased without reducing the other. In addition, the Pareto optimal linear classifier also supports the kernel tricks which yields much higher classification accuracy (Lanckriet, 2002; Kim, 2006).

2.5 Empirical Trade-off Analysis

The Pareto optimal linear classifier breaks down the cost-sensitive learning into three optimization problems in figure 1 that addresses three sections on the optimal trade-off curve respectively which are (0,1) to A, A to B, and B to (1,0) as shown in the top left image in figure 2 (Kim, 2006). The detailed formulation of the optimization problems for numerical computation is provided in Appendix A. With the optimal trade-off analysis, one can select the suitable classifier with specific true-positive and negative rates for particular use cases. Figure 2 demonstrates the comparison between the trade-off curves form Kim's paper and our reproduction of the empirical results. It shows that both our linear classifier and the kernel classifier have superior performance than the reference. This difference is difficult to explain as many details from the original implementation are unknown such as data preprocessing and numerical solver parameters. However, the overall trends of our results did align with the author's results.

$$\begin{aligned} \text{minimize} \quad & \sqrt{a^T\Sigma_+ a} + \lambda\sqrt{a^T\Sigma_- a} \\ \text{subject to} \quad & a^T(\mu_+ - \mu_-) = 1, \end{aligned}$$

$$\begin{aligned} \text{maximize} \quad & \beta \\ \text{subject to} \quad & \Phi\left(\frac{b - a^T\mu_-}{\sqrt{a^T\Sigma_- a}}\right) = \alpha, \\ & \Phi\left(\frac{a^T\mu_+ - b}{\sqrt{a^T\Sigma_+ a}}\right) = \beta, \end{aligned}$$

$$\begin{aligned} \text{maximize} \quad & \alpha \\ \text{subject to} \quad & \Phi\left(\frac{b - a^T\mu_-}{\sqrt{a^T\Sigma_- a}}\right) = \alpha, \\ & \Phi\left(\frac{a^T\mu_+ - b}{\sqrt{a^T\Sigma_+ a}}\right) = \beta, \end{aligned}$$

Figure 1: three optimization problems for Pareto optimal linear classification; left: compute section A to B on the optimal trade-off curve; middle: compute section (0,1) to A on the optimal trade-off curve; right: compute section B to (1,0) on the optimal trade-off curve; where a is the weight vector, b is the bias, μ_+, μ_- are mean of positive and negative classes, Σ_+, Σ_- are covariance matrix of positive and negative classes.

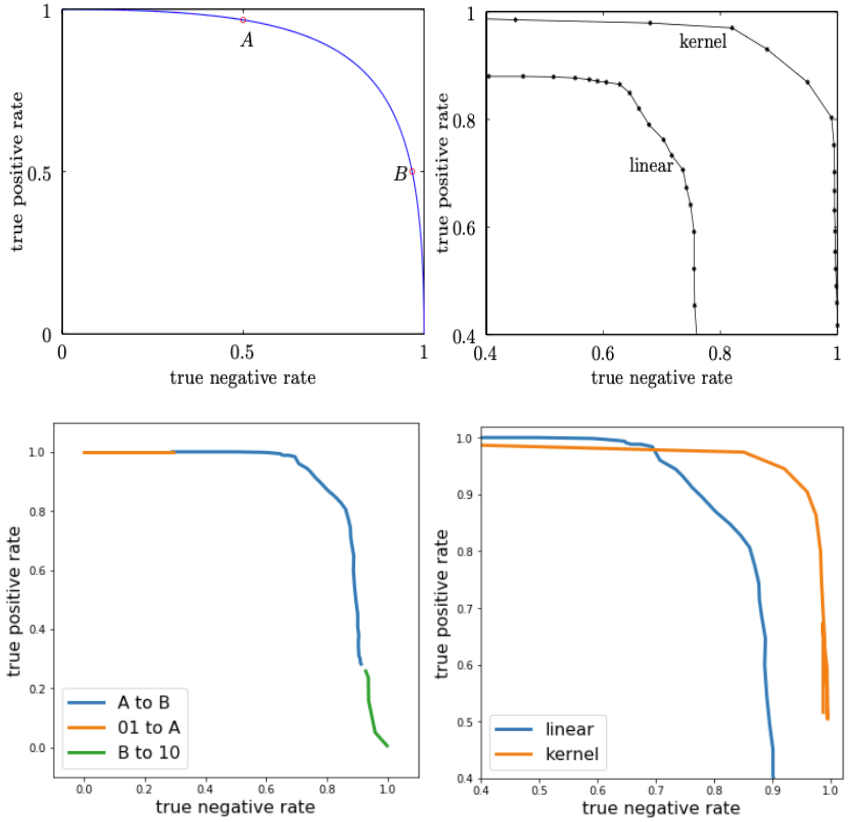


Figure 2: Empirical analysis was performed on the ionosphere benchmark data set with 70% training data and 30% test data. The data set consists of 34 features, 225 positive cases, and 126 negative cases. 20 trials with different train-test split were averaged to generate the final plots. Top left: Optimal trade-off curve with Gaussian class-conditional distribution (Lim, 2006); top right: original empirical optimal trade-off analysis results (Lim, 2006); bottom left: reproduced empirical optimal trade-off analysis results; bottom right: comparison of linear vs. kernel optimal trade-off analysis results.

Despite the high efficiency of the Pareto optimal linear classifiers, two problems exist for the current methods. First of all, the kernel method requires the computation of the kernel matrix, with parameter size equal to the number of samples. This is computational infeasible when the sample set is large (Bottou, 2007; List, 2009). The second problem is the asymmetry of the trade-off curve along the diagonal line, where the asymptote of the true positive rate is higher than that of the true negative rate in the empirical analysis. This may be due to the sample statistics as well as the unbalanced data set. In this report, these two problems are addressed by applying bagging and boosting to the original Pareto optimal linear classification methods.

3. Proposed Solution

3.1 Bagging

To address the problem of expensive computation of the kernel classifier with large data set, the concept of bagging is utilized. Bagging generates multiple predictors based on the bootstrap subsample set and aggregates the prediction from all predictors (Breiman, 1996). This method was originally purposed to mitigate the instability of some prediction methods such as decision trees. The fundamental idea of random forest is also built on top of bagging (Breiman, 2001). In this case, bagging is primarily used to reduce the size of the sample set while maintaining the same prediction accuracy as the single predictor scenario. Figure 3 shows the procedure for applying bagging to both optimal linear and optimal kernel classifiers.

Input:
Sequence of labeled data $< (x_1, y_1), \dots, (x_n, y_n) >$
Weak predictor (i.e. optimal linear or kernel classifier)
Integer T specifying number of weak predictors to be generated
Float $r \in (0,1)$ indicating the percentage of the bootstrap sub-dataset
Do for *t = 1, 2, ..., T*:
Create a weak predictor
Randomly generate a sub-dataset with parameter r
Solve corresponding optimization problem
Store obtained Pareto optimal classifier parameters
Prediction:
Do for *t = 1, 2, ..., T*:
Generate predictions using classifier t
Average all predictions in the end

Figure 3: Computational procedures for bagging wrapper over a weak classifier.

3.2 Modified Adaboost

To address the second problem which is the asymmetry of the optimal trade-off curve along the diagonal line during empirical analysis, a modified Adaboost algorithm is used. Adaboost is a type of boosting algorithm that aggregates prediction from multiple weak classifiers (Freund, 1997). The key difference between Adaboost and bagging is that Adaboost generates weak classifiers sequentially where the next classifier is fit to a modified dataset that is based on the performance of the previous classifier. Intuitively, the next modified dataset focuses more on the misclassified samples from the previous classifier by increasing their occurrence rate or sample weights. To address the asymmetry problem of the optimal trade-off curve, the key motivation is to readjust the weight of different classes such that they have a similar impact to the resultant classifier. Therefore, Adaboost is well suited for this purpose since it can adaptively modify the sample weights during the training process. Figure 4 shows the procedure for applying a modified Adaboost to the optimal linear classifiers. Unlike the bagging algorithm, the modified Adaboost algorithm does not scale to kernel methods due to the requirement of sample weight modification.

The procedure in figure 4 was modified based on the vanilla Adaboost algorithm (Freund, 1999). The major modification is done to the weight update function to include parameter λ. The vanilla version of Adaboost updates weights only based on the prediction correctness. This will conflict with the Pareto optimal linear classifiers as we have a specific target of true positive and true negative rates. For example, when λ is small, we favor true positive over true negative meaning that we want to correct false negative cases and ignore false positive cases. However, the Adaboost algorithm does not have access to this information and constantly tries to increase the weights for all false cases equally which will break the Pareto optimal condition. With the new update rule, the modified Adaboost algorithm adapts to what the Pareto optimal condition aims and changes sample weights accordingly.

Input:

Sequence of labeled data $< (x_1, y_1), \ldots, (x_n, y_n) >$
Weak predictor (i.e. Pareto optimal linear classifiers)
Integer *T* specifying number of weak learners to be generated
Float λ specifying the trade-off between true positive and true negative
Float *lr* specifying the learning rate

Initialize:

Weight vector ω^1 with dimension *n*

Do for *t = 1, 2, ..., T*:

Normalize weight vector

$$p^t = \frac{\omega^t}{\sum_{i=1}^{n} \omega_i^t}$$

Compute weighted $\mu_+, \mu_-, \Sigma_+, \Sigma_-$ for positive and negative cases with weights p^t
Obtain optimal linear classifier on weighted data via optimization
Compute prediction $\hat{y}^t \to [-1, 1]$
Compute error of prediction

$$\epsilon_t = \frac{1}{2} \sum_{i=1}^{n} p_i^t |\hat{y}_i^t - y_i|$$

Compute

$$\beta_t = lr \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right)$$

update new weight vectors

$$\gamma_t = \mathcal{I}\left(\frac{1}{2}(\hat{y}^t - y^t) < 0\right)\left(\frac{1}{1+\lambda}\right) + \mathcal{I}\left(\frac{1}{2}(\hat{y}^t - y^t) > 0\right)\left(\frac{\lambda}{1+\lambda}\right)$$

$$\omega^{t+1} \leftarrow \omega_i^t \exp(\beta_t \gamma_t)$$

Prediction:

$$\hat{y} = sign\left(\sum_{t=1}^{T} \beta_t y_t\right)$$

Figure 4: computational procedure for modified Adaboost wrapper over a weak classifier, where $\mathcal{I}$ is an indicator function that returns *1* when the statement is *True* and returns *0* when the statement is *False.* Input λ is identical to the λ from the optimization problem A to B in figure 1.

3.3 Sparse Optimal Linear Classifiers

One more necessary modification is to reformulate the optimization problem to encourage predictor sparsity. Both bagging and Adaboost algorithms were originally proposed to deal with weak classifiers such as decision stump or one-layer decision tree (Breiman, 1996; Freund, 1999). These weak classifiers usually have prediction accuracy slightly over 50%. In this case, the Pareto optimal linear classifier is not considered as weak as its prediction accuracy can easily achieve 80% on the Ionosphere data set. In the case of Adaboost, using high accuracy classifiers as weak learners will results in steep sample weight modification after each iteration, thus increase required number of weak learners for stable performance. To make the classifier weak, an additional constraint is added to the problem to ensure solution sparsity.

$$\begin{aligned} \underset{a}{\operatorname{argmin}} & \sqrt{a^T \Sigma_+ a} + \lambda \sqrt{a^T \Sigma_- a} \\ s.t. \quad & a^T(\mu_+ - \mu_-) = 1 \\ & a^T diag(\mathcal{V}) = 0 \leftarrow additional\ constraint \end{aligned}$$

The modified Pareto optimal linear classifier has additional sparsity constraints, where $\mathcal{V}$ is a random indication vector that contains 0 or 1. The index of value 1s indicates that the corresponding weights are 0. This classifier is used along with the bagging and modified Adaboost algorithm.

4. Experimental Results

4.1. Hyperparameter Tuning

To achieve optimal classification accuracy, the bagging method with the optimal linear classifiers requires tuning of three hyperparameters including the number of weak classifiers T, the percentage of the dataset r, and the percentage of the sparse entries for the classifiers u (percentage of 1s within $\mathcal{V}$). Although a simple grid search can be performed to select the global optimal hyperparameters in a 3-dimensional space, it is rather inefficient and computational heavy. To reduce the complexity of the hyperparameter search, the algorithm shown in figure 5 is implemented where an exact line search is performed along each hyperparameter with only one iteration. Although this procedure does not guarantee the global optimal solution, the resulting classifier performance should still perform close to optimal.

Initialize $r = 0.5$, $u = 0.5$
Do for $T=1,1+i, \ldots, 1+ni, i \in Z^+$:
  Evaluate optimal trade-off curve of classifier bagging (T, $r = 0.5$, $u = 0.5$)
  Find optimum T^*
Fix T^*
Do for $u = 1/j, 2/j, \ldots, j/j, j \in Z^+$:
  Evaluate optimal trade-off curve of classifier bagging (T^*, $r = 0.5$, u)
  Find the best u^*
While classifier performance does not degrade:
  $r \leftarrow r - 0.1$
  Evaluate optimal trade-off curve of classifier bagging (T^*, r, u^*)

Figure 5: Search procedure for bagging hyper parameter optimization.

Similarly, for the bagging method with kernel classifiers, I only optimize two parameters which are T and r as shown in Figure 6. The parameter u is not considered in the kernel classifier as the number of weight parameters is controlled by r. I perform the minimization of r first to reduce the computational complexity later on when optimizing for T.

Initialize $T = 5$
Do for $r = 1/j, 2/j, \ldots, j/j, j \in Z^+$:
  Evaluate optimal trade-off curve of classifier bagging ($T = 5$, r)
  Find the minimum r^* with no significant performance loss
Fix r^*
Do for $T=1,1+i, \ldots, 1+ni, i \in Z^+$:
  Evaluate optimal trade-off curve of classifier bagging (T, r^*)
  Find the best T^*

Figure 6: Search procedure for kernel bagging hyper parameter optimization.

The practical implementation of the modified Adaboost algorithm with the modified Pareto optimal linear classifier requires tuning of three hyperparameters including the number of weak classifiers T, value of learning rate lr, and the percentage of the sparse entries for the classifiers u. To simplify the searching problem, lr is set to a fixed value of 0.5. The resulting procedure is demonstrated in figure 7.

Initialize $u = 0.7$, $lr = 0.5$

Do for $T=1,1+i, \ldots, 1+ni, i \in Z^+$:

Evaluate optimal trade-off curve of classifier bagging (T, u = 0.7, lr = 0.5)
Find optimum T^*
Fix T^*
Do for u = $1/j$, $2/j$, ..., j/j, $j \in Z^+$:
Evaluate optimal trade-off curve of classifier bagging (T^*, u, lr = 0.5)
Find optimum u^*

Figure 7: Search procedure for modified Adaboost hyper parameter optimization.

For specific values of hyperparameters and empirical evidence for all three cases stated above, please see Appendix B.

4.2 Empirical Optimal Trade-off analysis

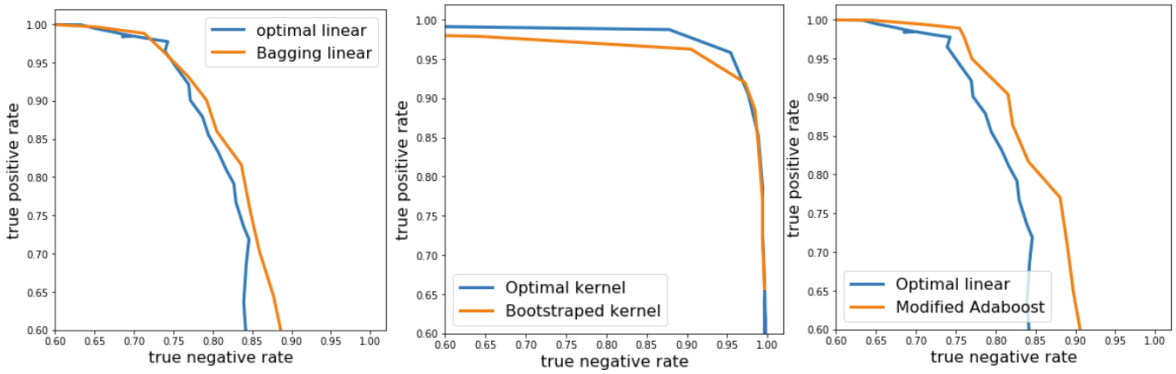


Figure 8: Empirical optimal trade-off analysis results for bagging and modified Adaboost algorithm; left: comparison between Pareto optimal linear classifier and itself with bagging; middle: comparison between Pareto optimal kernel classifier and itself with Gaussian kernel; right: comparison between Pareto optimal linear classifier and itself with modified Adaboost.

From the left image in figure 8, we showed that by applying bagging, the prediction accuracy of the Pareto optimal linear classifier is improved slightly as the optimal trade-off curve is shifted towards right. This is achieved by only using 50% of the original sample set for individual weak classifiers. We also showed that by applying bagging to optimal kernel classifier with only 30% of the original data set, the optimal trade-off curve remains similar performance with small performance degradation in true positive cases. The benefit of the reducing data set for individual weak classifiers did not show up in this case as the total sample set is small with 245 samples in the training set. However, computation for large data set will be significantly reduced due to the quadratic runtime of the kernel tricks (Bottou, 2007). Last, we compared the performance between optimal linear classifiers and itself with the modified Adaboost. Results showed that the modified Adaboost improved the prediction accuracy significantly in true negative cases which outperforms bagging method.

5. Conclusion

In this report, I addressed two problems with the Pareto optimal linear classification methods from Kim. To address the problem of the computational infeasibility of kernel-based Pareto optimal classifier, I implemented the bagging algorithm with sparse optimal linear classifiers. The empirical evidence showed that the bagging algorithm improved the linear classifier accuracy and maintained similar kernel classifier accuracy with the significantly reduced sample size for the individual sparse optimal linear classifiers. To address the problem of asymmetric optimal trade-off curves, I developed a modified Adaboost algorithm with sparse optimal linear classifiers. The empirical evidence showed further improvement in prediction accuracy over the bagging method.

I mention two future research directions. One is to introduce ensemble learning to the bagging algorithm where each weak learner is weighted in the final prediction stage. This is in spirit similar to EnsambleSVM (Claesen, 2014). The other is to introduce learning-based effective feature extraction methods such as AutoEncoder (Kramer, 1991). The autoencoder acts as a learnable kernel which only needs to be done once for a particular dataset and does not require computation of the kernel matrix during training or prediction.

6. Reference

Bach, F., Heckerman, D., & Horvitz, E. (2005). On the path to an ideal ROC curve: Considering cost asymmetry in learning classifiers. *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics,* 9–16.

Bottou, L., Lin, C. (2007) Support vector machine solvers. *Large Scale Kernel Machines, 301–320*. MIT Press.

Breiman, L. (1996). Bagging Predictors. *Machine Learning, 24,* 123-140.

Breiman, L. (2001). Random Forests. *Machine Learning, 45,* 5-32.

Chen, T. Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* 786-794.

Claesen, M., Smet, F., Suykens, Johan., Moor, B. (2014). EnsambleSVM: A Library for Ensemble Learning Using Support Vector Machines. *Journal of Machine Learning Research*, *15*, 141-145.

Cortes, C., Vapnik, V. (1995). Support-Vector Networks. *Machine Learning, 20,* 273-297.

Freund, Y., Schapire, R. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of computer and system sciences, 55,* 119-139.

Freund Y., Schapire, R. (1999). A Short Introduction to Boosting. *Journal of Japanese Society for Artificial Intelligence, 13(5)*, 771-780.

Kim, S., Magnani, A., Samar, M., Boyd, S. (2006), Pareto Optimal Linear Classification. *ICML 2006*, 473-480.

Kramer, M. (1991). Nonlinear Principal Component Analysis Using Autoassociative neural Networks. *AIChE Journal, 37*, No. 2.

Lanckriet, G., El Ghaoui, L., Bhattacharyya, C., & Jordan, M. (2002). A robust minimax approach to classification. *Journal of Machine Learning Research, 3,* 555–582.

List, N., Simon, H. (2009). SVM-optimization and steepest-descent line search. *COLT 2009*.

Rumelhart, D., Hinton, G., Williams, R. (1985). Learning Internal Representations by Error Propagation. *ICS Report 8506.*

Rumelhart, D., Hinton, G., Williams, R. (1986). Learning representations by back-propagating errors. *Nature, Vol 323.*

Rosenblatt, F. (1958), The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. *Psychological Review Vol. 65*, No. 6, 1958.

Appendix A - Optimization problems for generating optimal trade-off curves

Equation for the probability of true negative and positive cases where Φ is the CDF of standard normal distribution.

$$\begin{aligned}\Pr(a^T x < b \mid y = -1) &= \Phi\left(\frac{b - a^T\mu_-}{\sqrt{a^T\Sigma_- a}}\right)\\ \Pr(a^T x > b \mid y = +1) &= \Phi\left(\frac{a^T\mu_+ - b}{\sqrt{a^T\Sigma_+ a}}\right)\end{aligned}$$

Problem 1 - Section A to B in Figure 1 (left)

This problem applies to cases where:

$$Pr(a^T x < b|y = -1) \geqslant 0.5$$
$$Pr(a^T x > b|y = +1) \geqslant 0.5$$

Then, the optimization problem can be formulated as:

$$\underset{a}{argmin}\sqrt{a^T\Sigma_+ a} + \lambda\sqrt{a^T\Sigma_- a}$$
$$s.t.\ a^T(\mu_+ - \mu_-) = 1$$

The problem formulation can be further transformed such that it can be solved efficiently using CVX:

$$\underset{a}{argmin}\ ||Q_+ a||_2 + \lambda||Q_- a||_2$$
$$s.t.\ a^T(\mu_+ - \mu_-) = 1$$
$$where\ Q_+^T Q_+ = \Sigma_+ \quad Q_-^T Q_- = \Sigma_-$$

For generating the optimal trade-off curve, different values of λ from 0.01 to 100 are selected.

Problem 1 With Kernel Trick - Section A to B in Figure 1 (left)

When kernel trick is applied, the same optimization problem in problem 1 is done in kernel feature space.

$$\underset{\alpha}{argmin}\sqrt{\alpha^T F_+\alpha} + \lambda\sqrt{\alpha^T F_-\alpha}$$
$$s.t.\ \alpha^T G(g_+ - g_-) = 1$$

The problem formulation can be further transformed such that it can be solved efficiently using CVX:

$$\underset{\alpha}{argmin}\ ||P_+\alpha||_2 + \lambda||P_-\alpha||_2$$
$$s.t.\ \alpha^T G(g_+ - g_-) = 1$$
$$where\ P_+^T P_+ = F_+ \quad P_-^T P_- = F_-$$

where G is the kernel matrix. Refer to equation 13 in (Kim, 2006) for detailed expression for *F, G*, and *g*. For generating the optimal trade-off curve, different values of λ from 0.05 to 20 are selected.

Problem 2 - section (0,1) to A in Figure 1 (left)

Problem 2 applies to cases where true negative rate is below 0.5:

$$\underset{a}{argmax}\ Pr(a^T x > b|y = +1)$$
$$s.t.\ Pr(a^T x < b|y = -1) = \alpha,\ \alpha \in (0, 0.5)$$

The optimization problem can be formulated as a general constrained optimization problem:

$$\underset{a}{argmin}\left(-\frac{a^T\mu_+ - b}{\sqrt{a^T\Sigma_+ a}}\right)$$
$$s.t.\,\Phi\left(\frac{b - a^T\mu_-}{\sqrt{a^T\Sigma_- a}}\right) - \alpha = 0$$

For generating the optimal trade-off curve, different values of α from 0 to 0.5 are selected.

Problem 3 - section B to (1,0) in Figure 1 (left)

Similar to problem 2, problem three solves a similar problem with true positive rate below 0.5:

$$\underset{a}{argmax}\ Pr(a^T x < b | y = -1)$$
$$s.t.\ Pr(a^T > b | y = +1) = \beta,\ \beta \in (0, 0.5)$$

The optimization problem can be formulated as a general constrained optimization problem:

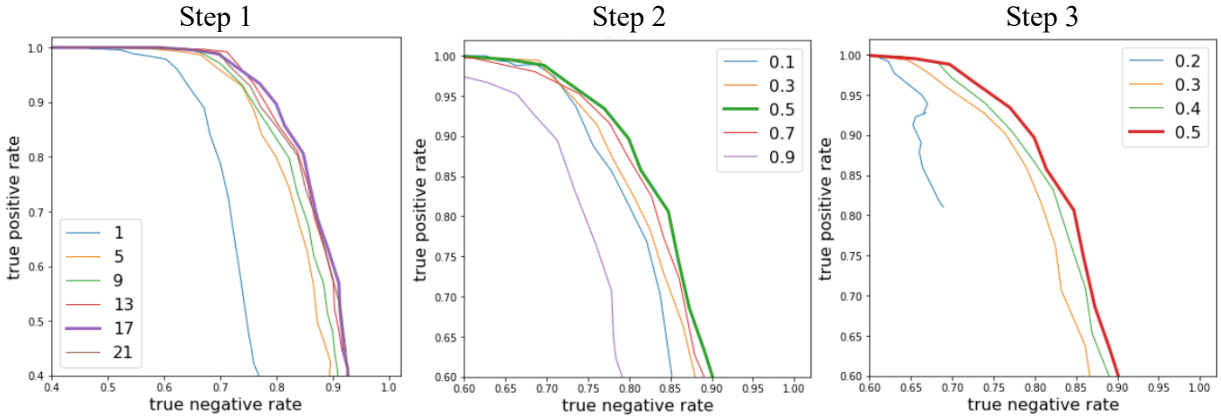
$$\underset{a}{argmin}\left(-\frac{b - a^T\mu_-}{\sqrt{a^T\Sigma_- a}}\right)$$
$$s.t.\,\Phi\left(\frac{a^T\mu_+ - b}{\sqrt{a^T\Sigma_+ a}}\right) - \beta = 0$$

For generating the optimal trade-off curve, different values of β from 0 to 0.5 are selected.

Appendix B – Hyperparameter Finding

Note: the train-test split in the final results generation is different from that of used in hyperparameter tuning. This is to avoid overfitting to the specific train-test splits.

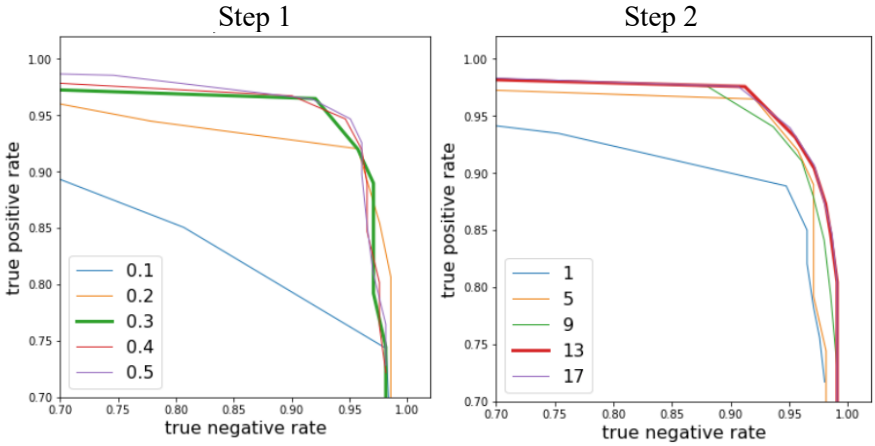
Bagging with Linear Classifier



Step 1: fix sparse ratio u = sample set size r = 0.5, vary number of weak classifiers T; get optimal T^* = 17
Step 2: fix T^* = 17, r = 0.5, vary u; get optimal u^* = 0.5
Step 3: fix T^* = 17, u^* = 0.5, vary r; get minimum r = 0.5

Note: each curve averages over 20 trials with different train-test split

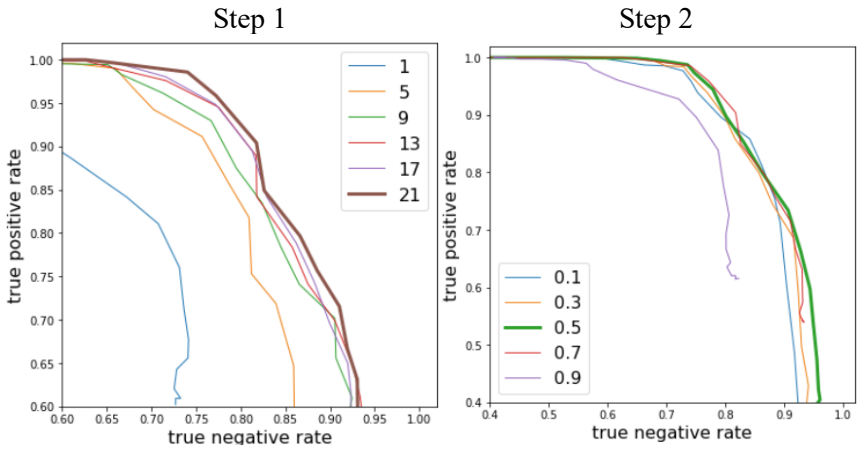
Bagging with Kernel Classifier



Step 1: fix $T = 5$, vary r; get minimum $r^* = 0.3$ without significant performance loss
Step 2: fix $r^* = 0.3$, vary T; get optimum $T^* = 13$

Note: each curve averages over 5 trials with different train-test split

Adaboost with Linear Classifier



Step 1: fix $u = 0.7$, $lr = 0.5$, vary T; get optimum $T^* = 21$
Step 2: fix $T^* = 21$, $lr = 0.5$, vary u; get optimum $u^* = 0.5$

Note: each curve averages over 10 trials with different train-test split