FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

ONLINE AND OFFLINE FEATURE SCREENING AND APPLICATIONS

By

MINGYUAN WANG

A Dissertation submitted to the
Department of Statistics
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

2021

Mingyuan Wang defended this dissertation on .
The members of the supervisory committee were:

Adrian Barbu

Professor Directing Dissertation

Piyush Kumar

University Representative

Yiyuan She

Committee Member

Jinfeng Zhang

Committee Member

The Graduate School has verified and approved the above-named committee members, and certifies that the dissertation has been approved in accordance with university requirements.

Dedicated to 26 years and beyond

ACKNOWLEDGMENTS

I would like to thank Dr. Adrian Barbu for his patient and endless support. He offered not only academic guidance, but also suggestions for life and career. I will never reach where I am without his dedication.

I also want to thank my committee members, Dr. Piyush Kumar, Dr. Yiyuan She and Dr. Jinfeng Zhang for their suggestions and time during the process of this dissertation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

The following short list of symbols are used throughout the document.

n	the number of observations
p	the number of variables
k	the number of true features
$S = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{R}, i = 1, ..., n\}$	the data space
X	the $n \times p$ data matrix
$X_j, j = 1, ..., p$	the j-th column/feature of X
$\mathbf{x}_i, i = 1, ..., n$	the i-th observation of X
$x_{ij}, i = 1, ..., nj = 1, ..., p$	the j-th column/feature of the i-th observation of X
$\mathbf{y}$	the $n \times 1$ target vector
$y_i, i = 1, ..., n$	the j-th target value

ABSTRACT

Filter or screening methods are often used as a preprocessing step for reducing the number of variables used by a learning algorithm in obtaining a classification or regression model. While there are many such filter methods, there is a need for an objective evaluation of these methods. Such an evaluation is needed to compare them with each other and also to answer whether they are at all useful, or a learning algorithm could do a better job without them. For this purpose, many popular screening methods are partnered in this thesis with three regression learners and five classification learners and evaluated on ten real datasets to obtain accuracy criteria such as R-square and area under the ROC curve (AUC). The obtained results are compared through curve plots and comparison tables in order to find out whether screening methods help improve the performance of learning algorithms and how they fare with each other. Our findings revealed that the screening methods were useful in improving the prediction of the best learner on two regression and two classification datasets out of the ten datasets evaluated.

While the screening methods show promising performance in case by case basis, modern data poses new challenges for existing methods that are designed to handle datasets with a limited magnitude. Nowadays datasets not only have higher dimension and larger sample size, but also have some unique characteristics that need to be taken into consideration. Signal data of different types, website information data, and others only exist for a short period of time and methods that only focus on dealing with high dimension and large sample size are not adequate to handle this type of data. Therefore a considerable number of online feature selection methods were introduced to handle these kind of problems in recent years. Online screening methods are one of the categories of online feature selection methods. They are used to preprocess data that is too large for batch screening methods to handle or to handle data that comes in sequential order and disappears soon after being processed. Furthermore due to the useful properties of the criteria of some screening methods such as mutual information and Gini index, some online screening methods are often integrated into online learning algorithms. Most online screening methods are concentrated on classification problems. Our research study focuses on classification as well. Researches are conducted to investigate whether online screening methods introduced in our research can obtain identical results as their offline counterparts. We are also interested to know how well our screening methods

with model adaptation can perform when facing data streams that have the concept drifting property. Experiments were conducted on real and synthetic classification datasets with binary labels. Results are summarized and analyzed in the form of comparison tables and comparison plots. We find that our online screening methods outperform their offline version in both computing time and space requirements. Furthermore, the results show that online screening methods with integrated model adaptation have a higher true feature detection rate than without model adaptation on data streams with the concept drifting property. Among the two large real datasets that potentially have the concept drifting property, online screening methods with model adaptation show advantages in either saving computing time and space, reducing model complexity, or improving prediction accuracy.

CHAPTER 1

INTRODUCTION

For the past few decades, with the rapid development of online social platforms and information collection technology, the concept of big data grew from a novel terminology in the past to one of the most powerful resources in present day. Especially in recent years, the sample sizes and feature dimensions of datasets rose to levels beyond precedent. This development poses great challenges for machine learning in extracting the relevant variables and in building accurate predictive models on such large datasets. Moreover, despite the fast development of computer hardware in recent decade. Big data still brings computation time and storage size issue for low budget setups.

One of the most popular machine learning tasks is feature selection, which consists of identifying fewer meaningful features (variables) from the original feature space, with the goal of reducing training speed and model size, as well as obtaining better prediction on unseen data, or obtaining better insight on the underlying mechanisms driving the target variable.

Feature selection methods have grown into a large family nowadays, with T-scoreDavis and Sampson [1986], Mutual Information [Lewis, 1992], Relief [Kira and Rendell, 1992], Lasso [Tibshirani, 1996], and MRMR [Han et al., 2005] as some of the more popular examples.

There are three categories of feature selection methods: screening methods (a.k.a. filter methods), wrapper methods and embedded methods. Screening methods are independent of the model learned. This makes them less computational complex. However for the same reason, screening methods tend to ignore more complex feature interaction. Wrapper methods use a learning algorithm (learner) to evaluate feature importance, which often leads to a better performance. But good performance comes with the possibility of overfitting, and a much higher computational demand. Embedded methods combine the feature selection and the model learning together, which generally makes them faster than the wrapper methods.

Throughout the years, these methods shined in various application frontiers. Given the exponential growth of data size, we were interested in evaluating the performance of existing screening

methods (one of the fastest) on conventional real high dimensional datasets and also going beyond that to introduce new screening methods to tackle bigger and more complex data.

1.1 Outline

In Chapter 2, we will go over various existing screening feature selection methods and evaluate their performance on real datasets. The screening methods will be used with different popular classification and regression learners. The comparison analysis will finally be carried on based on the performance metrics of each feature selection and learner combination. In Chapter 3, several online feature screening methods will be introduced to tackle large stream dataset with sparse inputs and concept drifting property. Experiments were conduct to compare their performance with their offline version, as well as on real world datasets.

CHAPTER 2

EMPIRICAL ANALYSIS OF OFFLINE SCREENING METHODS

2.1 Introduction

In this study, we focus on existing offline screening feature methods and would like an unbiased answer to the following questions:

- Do screening methods help to build good predictive models, or comparable models can be obtained without them?
- How do the existing screening methods compare with each other in terms of predictive capabilities, which one is the best and which one is the worst?

To answer these questions, we evaluated different screening methods (three for regression and seven for classification) on ten real datasets, five for regression and five for classification. The screening methods and the datasets will be described in the Methods section, but here we present our main findings.

The screening methods themselves cannot provide predictive models. For that purpose, different supervised learning algorithms such as SVM, Feature Selection with Annealing (FSA), Boosted Trees, and Naive Bayes were employed to construct the predictive models on the features selected by the screening methods.

For both regression and classification, experiments indicate that the screening methods are sometimes useful, in the sense they help obtaining better predictive models on some datasets. The findings are summarized in the comparison tables from the Results section.

Through our comparison study, we intend to provide researchers with a clear understanding of some of the well known screening (filter) methods and their performance of handling high-dimensional real data.

2.2 Related Work

The focus of this study is to examine the effect of screening (filter) methods on obtaining good predictive models on high-dimensional datasets. The recent literature contains several works that compare feature selection methods, including screening methods.

A recent feature selection survey [Li et al., 2017a] from Arizona State University (ASU) shows a comprehensive feature selection contents, studying feature selection methods from different data type perspectives. The survey is very broad, examining both supervised and unsupervised learning using binary and multi-class data, whereas our study focuses on supervised learning on regression and binary classification problems. The ASU study evaluates many classification datasets, but it does not have our goal of comparing feature screening methods and testing whether they are useful in practice or not. In this respect, we found some issues with the ASU study and we corrected them in this thesis. First, the ASU study uses the misclassification error as a measure of the predictive capability of a classifier. The misclassification error is sensitive to the choice of threshold, and is a more noisy measure than the AUC (area under the ROC curve). In our work we used the AUC instead, and obtained performance curves that have less noise, as it will be seen in experiments. Second, the ASU study obtains the results with 10-fold cross-validation, and are not averaged over multiple independent runs. In our work we used 7 independent runs of 7-fold cross-validation to further increase the power of our statistical tests. Third, we draw our comparisons and conclusions using statistical methods based on paired t-tests to obtain groups of similarly performing methods. An earlier version of the ASU report is [Tang et al., 2014], which is an overview of different types of feature selection methods for classification.

In [Chandrashekar and Sahin, 2014] are evaluated feature selection methods for flat features including filter methods, wrapper methods and embedded methods. However, tests are only conducted on low-dimensional datasets. In contrast we evaluate the filter methods on high dimensional datasets with 500-20,000 features and in many instances with more features than observations. Moreover, our goal is to compare filter methods themselves, not the filter-learning algorithm combination, since different datasets could have different algorithms that are appropriate (e.g. linear vs nonlinear). We achieve this goal by employing many learning algorithms and choosing the best one for each filter method and each dataset.

A comprehensive overview of the feature selection work done in recent years is shown in [Jović et al., 2015]. It covers feature selection methods including filter, wrapper, embedded and hybrid methods as well as structured and streaming feature selection. The article also discusses existing application of these feature selection methods in different fields such as text mining, image processing, industry and bioinformatics.

Recently [Cai et al., 2018] gave another detailed and broad overview of feature selection methods. The authors conducted their studies of many categories of feature selection methods, including but not limited to supervised, unsupervised, semi-supervised, online learning and deep learning. An experiment involving five feature selection methods was conducted on classification data. All five methods are either filter or wrapper methods. However they conducted their experiments on only two datasets, and the didn't consider the performance of the learning algorithms without any feature selection as a comparison baseline. Therefore the paper fails to show how much the feature selection methods could improve accuracy or whether they improved accuracy at all.

From a very interesting and unique standing point, [Li et al., 2017b] is an overview that focuses on the challenges currently facing feature selection research. They propose some solutions while at the same time reviewing existing feature selection methods. In [Urbanowicz et al., 2018] is evaluated the existing Relief method and some of its variants. The authors implemented and expanded these methods in an open source framework called ReBATE (Relief-Based Algorithm Training Environment). They described these methods in great detail and conducted simulation experiments with prior knowledge of the true features. They used a very vast simulated data pool with many varieties. The Relief variants were also compared with three other filter methods, using as performance measure the rate of detection of the true features. However, the paper didn't show if these methods can improve the performance of machine learning algorithms or if the improvement persists on real data.

Two other studies of feature selection methods are [Alelyani et al., 2013] and [Talavera, 2005]. In contrast to our study, they solely focus on unsupervised learning.

With the development of feature selection research, some well written feature selection software frameworks were also introduced. FeatureSelect [Masoudi-Sobhanzadeh et al., 2019] is a newly introduced such framework, which evaluated multiple trending feature selection methods on eight real datasets. Results were compared using various statistical measures such as accuracy, precision,

false positive rate and sensitivity. Their studies also evaluated five filter methods. Because the experiment didn't have learning algorithms without feature selection method as a benchmark, it again fails to show if using feature selection methods is better than not using them on these datasets. IFeatureChen et al. [2018] is another feature selection software framework dedicated to Python.

Some earlier studies also exist in this field (Guyon and Elisseeff, 2003Guyon and Elisseeff [2003], Sanchez-Marono et al.,2007 [Sánchez-Maroño et al., 2007], Saeys et al.,2007Saeys et al. [2007]).

Despite the large quantity of survey literature in screening feature selection methods, there lacks literature on evaluating their performance on real world datasets. We feel the necessity to fill this gap with a comprehensive performance analysis of screening methods using real world data.

2.3 Screening Methods for Classification

2.3.1 Mutual Information

The mutual information (a.k.a. information gain) method measures the information shared by two variables of interest, in this case, a feature X_j and the class label $\mathbf{y}$. The mutual information between variable A, where $S_A = \{A \in \mathbb{R}\}$ and variable Y, where $S_Y = \{Y \in \mathbb{R}\}$ can be described as:

$$I(A,Y) = \int_{S_A}\int_{S_Y} p(A,Y)\log\frac{p(A,Y)}{p(A)p(Y)}dAdY \tag{2.1}$$

where $p(A,Y)$ is the joint probability density of A and Y, while $p(A)$ and $p(Y)$ are the marginal p.d.f.s of A and Y.

In practice, given a sample dataset, each feature can be discretized into bins based on the value range. Here, $b = 1, 2, ..., B$ indicates bin number, $c = 1, 2, ..., C$ indicates class number. Therefore mutual information between label vector $\mathbf{y}$ and feature vector X_j can also be described as:

$$I(X_j,\mathbf{y}) = \sum_{b=1}^{B}\sum_{c=1}^{C} p(X_{j_b},\mathbf{y}_c)\log\frac{p(X_{j_b},\mathbf{y}_c)}{p(X_{j_b})p(\mathbf{y}_c)} \tag{2.2}$$

where $p(X_{j_b},\mathbf{y}_c)$ is the joint probability of bin X_{j_b} and label vector $\mathbf{y}_c$, while $p(X_{j_b})$ and $p(\mathbf{y}_c)$ are the marginal probabilities. Features that are more related to the classification label tend to have higher mutual information.

2.3.2 Relief and ReliefF

The idea of the Relief algorithm is to measure how well a feature's values can distinguish instances that are near each other. For the i-th instance-label pair $(\mathbf{x}_i, y_i)$, denote its nearest instance neighbor from the same class as the nearest hit $(\mathbf{x}_i^{hit}, y_i)$, and its nearest instance neighbor from a different class as the nearest miss $(\mathbf{x}_i^{miss}, y_i^{miss})$. The distance between two instances $\mathbf{x}_i, \mathbf{x}_j$ is calculated using the Euclidean norm $\|\mathbf{x}_i - \mathbf{x}_j\|$. Then the Relief measure for a certain feature F can be computed as:

$$Relief_j = \frac{1}{n}\sum_{i=1}^{n}[\text{diff}(F : x_i, x_i^{miss}) - \text{diff}(F : x_i, x_i^{hit})] \tag{2.3}$$

where the function $\text{diff}(F : x, y)$ calculates the difference between the values of feature F for two instances. For discrete features $\text{diff}(F : x, y)$ is defined as:

$$\text{diff}(F : x, y) = \begin{cases} 0; & \text{if } x = y \\ 1; & \text{otherwise} \end{cases} \tag{2.4}$$

and for a continuous feature X_j as:

$$\text{diff}(F : x, y) = \frac{|x - y|}{\max(F) - \min(F)} \tag{2.5}$$

The Relief measure can also be extended to a multi-class version ReliefF, but we are only interested in binary classification in this thesis.

In summary, higher Relief values indicate better discrimination power of the label by the feature values.

2.3.3 Minimum Redundancy Maximum Relevance

The minimum redundancy maximum relevance (MRMR) method is set to choose the feature that has the highest mutual information difference (MID) or mutual information quotient (MIQ). The MID and MIQ are calculated as :

$$MID_j = I(X_j, \mathbf{y}) - \frac{1}{|Q|}\sum_{q \in Q} I(X_j, X_q) \tag{2.6}$$

$$MIQ_j = \frac{I(X_j, \mathbf{y})}{\frac{1}{|Q|}\sum_{q \in Q} I(X_j, X_q)} \tag{2.7}$$

where Q is the set of features already selected, $I(X_j, \mathbf{y})$ is the mutual information for j-th feature and the label vector $\mathbf{y}$, and $I(X_j, X_q)$ denotes the mutual information between features j and q.

In the case where the features take continuous values, MIQ and MID can be modified as the F-test correlation difference (FCD) and F-test correlation quotient (FCQ). FCD and FCQ are computed as:

$$FCD_j = F(X_j, \mathbf{y}) - \frac{1}{|Q|}\sum_{q \in Q} |c(X_j, X_q)| \tag{2.8}$$

$$FCQ_j = \frac{F(X_j, \mathbf{y})}{\frac{1}{|Q|}\sum_{q \in Q} |c(X_j, X_q)|} \tag{2.9}$$

where $F(X_j, \mathbf{y})$ is the F-statistic for j-th feature and label vector $\mathbf{y}$, and $|c(X_j, X_q)|$ denotes the absolute correlation coefficient between features j and q. In the case of binary labels the F-statistic can be replaced by the T-statistic.

2.3.4 T-Score

The T-score method is a feature screening method applied on datasets with binary labels. The method is based on the calculation of the t-statistic. The basic idea is to divide each feature's values into two sample groups based on their labels. Then the t-statistic is calculated to examine if the two sample groups have statistically significant differences in their means. For each feature X_j, the values of X_j are divided into two groups based on their labels. Then the means μ_1 and μ_2 are calculated as the means of the two groups and σ_1 and σ_2 are standard deviations of these two groups respectively. Let n_1 and n_2 be the number of instances of the two groups. Then the t-statistic for feature i can be calculated as:

$$T_j = \frac{|\mu_1 - \mu_2|}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}} \tag{2.10}$$

Generally speaking, the higher the t-statistic, the more separated the two labels are by values of that feature and therefore the more relevant that feature is for classification.

2.3.5 Chi-square Score

The chi-square score method is based on the chi-square statistic. It can test the independence between two variables, therefore it can also test the relevance of a variable X_j for the label vector $\mathbf{y}$. If feature X_j has L levels (discretized if necessary) and $\mathbf{y}$ has C=2 levels (label categories), let n_{lc} denote the number of instances with label c and level l for feature j. Let $\hat{n}_{lc}$ denote the estimated number of instances with label c and having level l, $\hat{n}_{lc} = \frac{n_l n_c}{n}$, where n is the total number of instances, n_l is the number of instances having level l, and n_c is the number of instances with label c. The chi-square statistic is then computed as:

$$\chi_j^2 = \sum_{l=1}^{L} \sum_{c=1}^{C} \frac{(n_{lc} - \hat{n}_{lc})^2}{\hat{n}_{lc}} \tag{2.11}$$

Usually, a higher chi-square statistic indicates low independence, in other word, a higher relevance between that feature and label.

2.3.6 Gini Index

The Gini index method is based on the Gini impurity after splitting a sample set. For a given feature X_j, let $A_h = \{i, x_{ij} \leq h\}$ denote the instances whose values of the j-th feature is smaller than or equal to h and $B_h = \{i, x_{ij} > h\}$. The Gini impurity for subset A_h or B_h can be expressed as:

$$Gini(A_h) = 1 - \sum_{c=1}^{C} P(C_c|A_h)^2 \tag{2.12}$$

where C is the number of labels and $c \in \{1, 2, ..., C\}$ are the label categories. $P(C_c|A_h)$ is the conditional probability of instances having label c given that they are in subset A_h. Let a_c denote the number of instances in A_h with label c. Let a_h denote the number of instances in A_h. Then $P(C_c|A_h)$ can be calculated as a_c/a_h.

Based on these notations, the Gini index after splitting is:

$$Gini_{split} = P(A_h)Gini(A_h) + P(B_h)Gini(B_h) \tag{2.13}$$

where $P(A_h)$ is the number of instances in subset A_h divided by the number of total instances. Therefore for each feature, the Gini index can be calculated as:

$$Gini_j = P(A_h)(1 - \sum_{c=1}^{C} P(C_c|A_h)^2) + P(B_h)(1 - \sum_{c=1}^{C} P(C_c|B_h)^2) \tag{2.14}$$

Basically, the Gini index measures the frequency that a randomly chosen instance from the sample set would be incorrectly labeled. So for all possible thresholds h of one feature, select the minimum Gini index as this feature's Gini index. Features with smaller Gini index are preferred.

2.3.7 Fisher Score

The idea of the Fisher score is to choose the feature subset, for which the observations have the largest possible between class distances and the smallest possible within class distances. This would be the feature subset that has the largest Fisher score. The Fisher score for any feature set is computed as:

$$Fisher = Tr(D_b)(D_t + \gamma I)^{-1} \tag{2.15}$$

where γ is a regularization term, D_b is called between-class scatter matrix, D_t is called total scatter matrix. Since for a certain feature subset with size d, there are $\binom{m}{d}$ combinations of Fisher scores to be calculated, this is too computationally expensive. For this reason, a heuristic is to compute the scores for each feature with respect to the Fisher score criterion. The individual Fisher score is computed as:

$$Fisher_j = \frac{\sum_{c=1}^{C} n_c(\mu_c - \mu)^2}{\sum_{c=1}^{C} n_c \sigma_c^2} \tag{2.16}$$

where μ and σ are mean and standard deviation of that feature, and μ_c is the mean of the feature values for observations with label c and n_c is the number of instances with label c. Features with larger Fisher scores are preferred.

2.4 Screening Methods for Regression

2.4.1 Correlation

The correlation feature screening method is based on the calculation of correlation coefficient between response and features. It is evaluated as following:

$$\rho_j = \frac{cov(X_j, \mathbf{y})}{\sigma_{\mathbf{y}} \sigma_{X_j}} \tag{2.17}$$

where X_j is j-th feature, $\mathbf{y}$ is response. Features with larger correlation coefficient are preferred.

2.4.2 Mutual Information

To apply mutual information for regression data, we discretized both the feature and the response into a numbers of bins. For feature X_j and response $\mathbf{y}$, let x_{jb} and y_l indicate values falling in b-th and l-th bins respectively.

The mutual information for the j-th feature is computed as:

$$I(X_j, Y) = \sum_{b=1}^{B} \sum_{l=1}^{L} P(x_{jb}, y_l) \log \frac{P(x_{jb}, y_l)}{P(x_{jb})P(y_l)} \tag{2.18}$$

Let n denote the number of instances. Then $P(x_{jb}, y_l)$ can be estimated by N_{jbl}/n, where N_{jkl} is the number of instances falling into feature bin b and response bin l. Also, $P(x_{jb})$ can be estimated by N_{jb}/n, where N_{jb} is the number of instances lay in feature bin b, and $P(y_l)$ can be estimated by N_l/n, where N_l is the number of instances lay in response bin l. Features with larger mutual information have more influence on the response.

2.4.3 RReliefF

RReliefF is a regression version of Relief. It starts from the original weight function. For feature A the function can be expressed as:

$$\begin{aligned} W(A) = &P(\text{different value of A}|\text{nearest instance from different class}) \\ &-P(\text{different value of A}|\text{nearest instance from the same class}) \end{aligned} \tag{2.19}$$

Denote

$$\begin{aligned} P_{diffA} &= P(\text{different value of A}|\text{nearest instances}) \\ P_{diffP} &= P(\text{different response}|\text{nearest instances}) \\ P_{diffP|diffA} &= P(\text{different response}|\text{different value of A and nearest instances}). \end{aligned} \tag{2.20}$$

Then from (2.19), using Bayes' rule:

$$W(A) = \frac{P_{diffP|diffA} P_{diffA}}{P_{diffP}} - \frac{(1 - P_{diffP|diffA}) P_{diffA}}{1 - P_{diffP}}, \tag{2.21}$$

which can be further modified as:

$$W(A) = \frac{N_{dP\&dA}}{N_{dP}} - \frac{(N_{dA} - N_{dP\&dA})}{m - N_{dP}} \tag{2.22}$$

where N_{dA}, N_{dP} and $N_{dP\&dA}$ denote different feature value, different response value and different feature & response value respectively. Denote for instance $\mathbf{x}_i$ its k-nearest instances as $\mathbf{u}_{ij}, j \in \{1, ..., k\}$. Then the expressions for N_{dA}, N_{dP} and $N_{dP\&dA}$ are:

$$\begin{aligned} N_{dA} &= \sum_{i=1}^{n}\sum_{j=1}^{k} \text{diff}(A : \mathbf{x}_i, \mathbf{u}_{ij}) d(i,j) \\ N_{dP} &= \sum_{i=1}^{n}\sum_{j=1}^{k} \text{diff}(y : \mathbf{x}_i, \mathbf{u}_{ij}) d(i,j) \\ N_{dP\&dA} &= \sum_{i=1}^{n}\sum_{j=1}^{k} \text{diff}(y : \mathbf{x}_i, \mathbf{u}_{ij}) \,\text{diff}(A : \mathbf{x}_i, \mathbf{u}_{ij}) d(i,j) \end{aligned} \tag{2.23}$$

Where $\text{diff}(F, x, y)$ is defined in Eq. (2.4) and (2.5) and $d(i,j)$ is used to take account the distance between $\mathbf{x}_i$ and $\mathbf{u}_j$:

$$d(i,j) = \frac{d_1(i,j)}{\sum_{l=1}^{k} d_1(i,l)} \tag{2.24}$$

and

$$d_1(i,j) = \exp(-\text{rank}^2(\mathbf{x}_i, \mathbf{u}_{ij})/\sigma^2) \tag{2.25}$$

where $\text{rank}(\mathbf{x}_i, \mathbf{u}_{ij})$ is the rank of the instance $\mathbf{u}_{ij}$ in a sequence of instances ordered by the distance from $\mathbf{x}_i$, and σ is a user defined parameter. $d_1(i,j)$ is calculated in an exponentially decreasing fashion with the idea that further instances should have lesser influence. Usually, $d_1(i,j)$ takes value $1/k$. Features with larger $W(\cdot)$ are preferred.

2.5 Feature Selection With Annealing (FSA)

Feature Selection With Annealing (a.k.a. FSA) is a recent embedded method for feature selection that can handle high dimensional data. FSA can bring the relevant feature space down to an acceptable level using an variable removal schedule and obtain a rather accurate and stable model. The basic algorithm of FSA is:

The value of N^{iter} in step 2 is the total number of iterations. The formula in step 3 uses a typical gradient descent or an epoch of stochastic gradient descent with momentum and minibatch towards minimizing the loss $L(\boldsymbol{\beta})$. The M_e in step 4 is the annealing schedule which gradually decreases with the iteration number e. It decides how many features to keep in each iteration. Let

Algorithm 1 Feature Selection with Annealing (FSA)

Input: Training samples $(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{R}, i = 1, 2, ..., N$.

Output: Trained model parameter vector $\boldsymbol{\beta}$.

1: Initialize $\boldsymbol{\beta}$.
2: **for** e=1 to N^{iter} **do**
3: Update $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \eta \frac{\partial L(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}$
4: Keep the M_e features with highest $|\boldsymbol{\beta}_j|$ and renumber them 1, ..., M_e.
5: **end for**

k be a user defined parameter controlling how many features to keep in the end. The M_e can be computed as:

$$M_e = k + (p - k) \max(0, \frac{N^{iter} - 2e}{2e\mu + N^{iter}}), e = 1, ..., N^{iter} \tag{2.26}$$

where p is the feature dimension of the original input data and μ is the annealing parameter which can be tuned using cross validation. FSA has good computational efficiency and theoretical guarantees of consistency. The user defined parameter k denoting how many features to select is more intuitive than the penalty parameter in the penalized methods (e.g. L_1 penalized regression) and makes the procedure more controllable. Experiments were conducted separately for regression and classification. For regression, the screening methods were Correlation, Mutual Information [Lewis, 1992], and RReliefF [Robnik-Šikonja and Kononenko, 1997]. These screening methods were combined with learners including Feature Selection with Annealing (FSA) [Barbu et al., 2017], Ridge Regression, and Boosted Regression Trees.

For classification, the screening methods were T-score [Davis and Sampson, 1986], Mutual Information [Lewis, 1992], Relief [Kira and Rendell, 1992], Minimum Redundancy Maximum Relevance (MRMR) [Han et al., 2005], Chi-square score [Liu and Setiono, 1995], Fisher score [Duda et al., 2012], and Gini index [Gini, 1912]. They were combined with learners including FSA [Barbu et al., 2017], Logistic Regression, Naive Bayes, SVM, and Boosted Decision Trees.

Among these screening methods, Mutual information, Correlation, Gini index, Fisher-score, Chi-square score and T-score select features individually. In contrast, MRMR requires to calculate the redundancy between the already selected features and the current feature, and Relief requires to calculate the distance between two observations using Euclidean norm so that one can determine the nearest neighbor with the same label and with different labels. The calculation of Euclidean

norm involves all the feature value. Consequently, these two methods select features in combination and are slower than the other methods.

2.6 Evaluation of Screening Methods

The predictors of all datasets were normalized to zero mean and standard deviation 1 in a pre-processing step. For each dataset, experimental results were obtained as the average of 7 independent runs of 7-fold cross-validation. For each run, a random permutation of the dataset was generated and the data was split into seven approximately equal subsets according to the permutation. Then a standard full 7-fold cross-validation was performed as follows. Each fold consists of testing on one of the subsets after training on the other six. This procedure was run with each of the seven subsets as the test set and the other six as the training set. For each fold, each one of the screening methods mentioned above was used to reduce the dimension of the feature space to the desired size, then a learning algorithm using preset parameter values was applied on the selected features to obtain the model. The predictions of the model on the test subset for each fold were combined to obtain a vector of test predictions on the entire dataset, which was used to obtain performance measures (R^2 for regression and AUC (Area under the ROC curve) for classification). To increase accuracy, these performance measures were averaged over seven independent runs on different permutations of the data.

To insure the consistency of the comparison, the number of features that were selected by each screening method was kept the same for each dataset. For each dataset (except Wikiface), 30 different values of the number of selected features were assigned. Plots were used to compare the average performance over the 7 cross-validated runs of different combinations of screening method and learner. Also for each combination, the optimal number of selected features was selected based on the maximum average test performance over the 7 cross-validated runs. Pairwise t-tests at the significance level α=0.05 were used to compare between different combinations to see if they are significantly different.

2.7 Construction Tables of Groups

Groups of screening method-learner combinations that are not significantly different from each other were constructed as follows (we use paired t-tests to obtain p-values when comparing differ-

ent methods combinations and set 0.05 as the significance level in our experiment). The screening method-learner combinations are first sorted in descending order of their peak performance. Then starting from the first combination F downward, the last combination in the sequence that is not statistically significantly different from combination F is marked as combination L. All combinations between F and L are put into the same group. The same procedure was used for other combinations along the sequence. All these tables of groups are provided in the Supporting information section.

2.8 Construction Comparison Tables

Comparison tables were established based on how many times each screening method - learner combination appeared of in the group tables. Three kinds of counting methods were applied.

1) *The number of datasets where the screening method performed significantly better than no screening* for different learning algorithms. For each learning algorithm, it is the number of datasets on which the screening method appeared in higher group tiers than the same learning algorithm without screening. This counting method is used to construct Table 2.2 and Table 2.6 except the "Best" column.

2) *The number of datasets where the screening method was significantly better than the best performing algorithm with no screening (usefulness per dataset).* For each dataset, we checked for each screening method whether it appeared with a learning algorithm in a higher group tier than the best learning algorithm without screening. This counting method is used to construct Table 2.3 and Table 2.7 and the "Best" column of Tables 2.2 and 2.6. Column with name "Total Count" is generated from the summation of counts across all datasets for each screening methods.

3) *The number of datasets where each filter-learning algorithm combination was in the top performing group (top performing).* This counting method is used in Table 2.4 and Table 2.8.

2.9 Real Data Analysis

2.9.1 Data sets

Five datasets were used for regression and five datasets for classification, with the specific dataset details given in Table 2.1.

Table 2.1: The datasets used for evaluating the screening methods. The parameter τ controls the number of selected features as $[(4t)^\tau], t = \overline{1, 30}$.

Dataset	Learning type	Feature type	Number of features	Number of observations	τ
Mouse BMI [Wang et al., 2006]	Regression	Continuous	21575	294	1.825
Tumor [Grossman et al., 2016]	Regression	Continuous	16790	1750	1.825
Indoorloc [Torres-Sospedra et al., 2014]	Regression	Continuous	520	20294	1.25
Wikiface [Rothe et al., 2016]	Regression	Continuous	4096	53040	1.65
CoEPrA2006 [Ivanciuc, 2006]	Regression	Continuous	5787	133	1.68
Gisette [Guyon et al., 2005]	Binary Classification	Continuous	5000	7000	1.73
Dexter [Guyon et al., 2005]	Binary Classification	Continuous	20000	600	1.78
Madelon [Guyon et al., 2005]	Binary Classification	Continuous	500	2600	1.25
SMK_CAN_187 [Spira et al., 2007]	Binary Classification	Continuous	19993	187	1.78
GLI_85 [Freije et al., 2004]	Binary Classification	Continuous	22283	85	1.78

The regression dataset Indoorloc is available on the UCI Machine Learning Repository [Lichman, 2013]. The original dataset has eight indicator columns including longitude, latitude and so on. In our study, we only used the latitude as response. We combined the training and validation data files and deleted all duplicated observations due to the removing of the other seven indicator columns. The dataset Tumor was extracted from TCGA (The Cancer Genome Atlas). The response of this dataset is the survival time(in days) of the patient, and the predictors represent gene expression levels. The classification datasets Gisette, Dexter, Madelon are part of the NIPS 2003 Feature selection challenge [Guyon et al., 2005] and are also available on the UCI Machine Learning Repository.

The dataset Wikiface is a regression problem of predicting the age of a person based on the person's face image, and was obtained from the Wikiface images [Rothe et al., 2016]. A CNN (Convolutional Neural Network) vgg-face [Parkhi et al., 2015] pre-trained for face recognition was applied to each face and the output of the 34-th layer was used to generate a 4096 feature vector for each face. This 4096 dimensional vector was used as the feature vector for age regression, with the age value from the original Wikiface data as the response.

2.9.2 Regression Results

The following results are based on the output generated using Matlab 2016b [Mat, 2016]. For RReliefF, correlation score, ridge regression and boosted regression trees we used their Matlab 2016b implementation. Mutual information for regression was implemented by ourselves. For FSA we used the Github[1] implementation from its original authors.

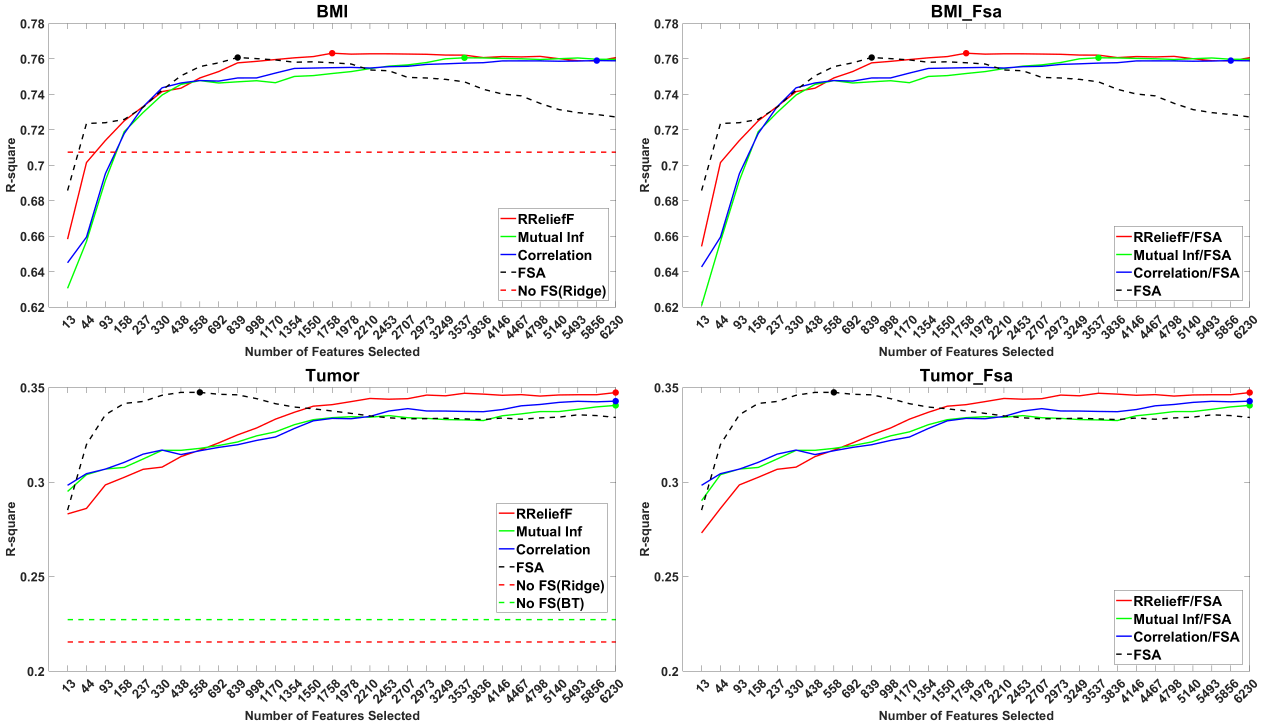

Figure 2.1: Performance plots of methods with and without feature screening. Left: for each screening method are shown the maximum R^2 value across all learners. Right: R^2 of the screening methods with the best learner for this data (FSA).

Performance Plots. For the regression datasets, the plots from Fig. 2.1 and 2.2 show the R^2 value vs. the number M_i of selected features, where $M_i = [(4i)^\tau], i = 1, ..., 30$. The value of τ for each dataset is given in Table 2.1.

In Fig 2.1, left, are shown the R^2 of the best learning algorithm vs. the number of features selected by a screening method for the BMI and tumor datasets. Observe that these datasets are both gene expression datasets with many features and few observations. In Fig 2.1, right, are shown the R^2 of FSA (the best overall learning algorithm) vs. the number of features selected by a screening method. Except a slightly higher value given by RReliefF on the BMI data, overall the

[1] `https://github.com/barbua/FSA`

screening methods did not show higher scores than that of the optimal regression learners for the BMI and Tumor datasets. The plots on the right show that the screening methods even needed to select more features to obtain similar performance to FSA without screening.

In Fig 2.2, left, are shown the R^2 of the best learning algorithm vs. the number of features selected by a screening method for the other three regression datasets. In Fig 2.2, right, are shown the R^2 of the best overall learning algorithm in each case (ridge for CoEPrA and Wikiface, boosted trees for Indoorloc) vs. the number of features selected by a screening method. From the plots we observe that screening methods give slightly better results than the learning algorithms without screening on the Indoorloc and Wikiface datasets. The statistical significance of the improvement can be seen in the table of groups from the supporting information or in the comparison tables below.

Comparison Tables. The counts in the comparison table are based on the table of groups from the supporting information.

Table 2.2: Overview of the number of datasets where each feature screening method performed significantly better than no screening for different learning algorithms and than the best performing algorithm (larger numbers are better).

Screening Method	FSA	Ridge	Boost Tree	Best algorithm
RReliefF [Robnik-Šikonja and Kononenko, 1997]	0	4	3	2
Mutual Information [Lewis, 1992]	0	3	3	2
Correlation	0	4	3	1

In Table 2.2 is shown the number of datasets where a filter method helps an algorithm perform significantly better, and the number of datasets where a screening method significantly improves a learning algorithm compared to the best performing learning algorithm without screening. It is shown that screening methods have relatively good performance with ridge regression and boosted regression trees on datasets tested. They work on 3-4 out of 5 datasets. RReliefF method and Mutual information method have slightly better performance than Correlation method when only comparing with the best learner without screening methods.

In Table 2.3 is shown a "*" for each dataset and each screening method when it has a learning algorithm that obtains significantly better performance than the best learning algorithm without screening. An "=" sign shows for each dataset when the screening method is in the same performance group as the best learning algorithm without screening method (so it does no harm).

Table 2.3: Ranking of feature screening methods for regression by number of datasets where screening method was significantly better than the best performing no screening method. (larger numbers are better)

Screening Method	BMI	Tumor	CoEPrA	Indoorloc	Wikiface	Total Count
RReliefF	=	=	=	*	*	2
Mutual Information.	=			*	*	2
Correlation	=		=	=	*	1

We can see that Mutual Information and RReliefF worked on the Indoorloc dataset and all three screening methods worked on the Wikiface data. However, the screening methods didn't provide performance improvement on the other three regression datasets. It is also shown that only in few occasions that screening methods harm the best learning algorithm. This is shown by the blank cells in table.

Table 2.4: Number of datasets each combination was in the top performing group.

Filter \ Learners	FSA	Ridge	Boost Tree
RReliefF	3	0	1
Mutual Information	1	1	1
Correlation	2	1	0
—	3	0	0

In Table 2.4 is shown the counts of screening method-learner combinations that are in the top group. The combination of FSA with screening methods worked on more regression datasets than the other screening-learner combinations. However it was not a significant improvement compared to FSA without screening, which also worked on 3 out of 5 datasets.

Table 2.5: Ranking of feature screening methods for regression by the number of times each was in the top performing group. (larger numbers are better)

Screening Method	Top performing Method-Algorithm	Method
RReliefF	4	4
Mutual Information	3	3
Correlation	3	2
No Screening	3	3

In Table 2.5 is shown the number of times each screening method was in the top performing group. In the first column, these methods were counted together with the learning algorithms they were applied. So there can be at most 15 counts (For each screening method there are three learning algorithms and five datasets total) in each cells. The second column shows the counts withe the best learning algorithm for each method, so there can be at most 5 counts in each cell. The table shows that RReliefF has the best performance, which is larger that the worst performance (correlation) by 2. Among the three screening methods only RReliefF has a higher count than non screening.

2.9.3 Classification Results

The following results are based on the output generated by Matlab 2016b. For the methods Relief, T-score, chi-square score, logistic regression, naive Bayes, SVM, boosted decision trees we used their Matlab 2016b implementation. For MRMR, Fisher score, and Gini index we used the ASU repository implementation[2]. Mutual information for classification was implemented by ourselves. Some of the implementations only accept discrete predictors, so the quantile-based discretization method [Nguyen, 2014] was used.

Performance Plots. In Fig 2.3, left are shown the AUC of the best learning algorithm vs. the number of features selected by a screening method for four of the classification datasets. In Fig 2.3, right, are shown the AUC of the best overall learning algorithm in each case (SVM for SMK_CAN_187, Boosted trees for Madelon and Dexter, FSA for Gisette) vs. the number of features selected by a screening method.

The plots show that all screening methods help obtain better results on the Gisette and Madelon datasets and most screening methods help obtain better results on the SMK_CAN_187 data. It can be observed that on the SMK_CAN_187 and Madelon, although some screening methods show better results, they select a higher number of selected features than FSA when they reach their optimal values. The right side figure of the SMK_CAN_187 plots shows that Relief doesn't work well with the best learner for this dataset. Only three out of the seven methods help obtain a better result on the Dexter dataset.

In Fig 2.4, left are shown the AUC of the best learning algorithm vs. the number of features selected by a screening method on the GLI_85 dataset. In Fig 2.4, right, are shown the AUC of

[2] `http://featureselection.asu.edu/old/software.php`

FSA (for GLI_85) vs. the number of features selected by a screening method. We can observe that only one screening method (mutual information) helps obtain better results than the best learning algorithm without screening.

Table 2.6: Overview of the number of datasets where each feature screening method performed significantly better than no screening for different learning algorithms and than the best performing algorithm (larger numbers are better).

Screening Method	Boost Tree	FSA	SVM	NB	Logistic	Best algorithm
Mutual Information	2	1	3	5	5	2
Fisher Score [Duda et al., 2012]	3	1	2	5	5	2
Chi-square Score [Liu and Setiono, 1995]	2	1	2	5	4	2
Gini Index [Gini, 1912]	2	1	2	5	4	2
Relief [Kira and Rendell, 1992]	3	2	1	5	4	1
T-score [Davis and Sampson, 1986]	2	1	2	5	5	2
MRMR [Han et al., 2005]	2	1	2	5	4	2

Comparison Tables. In Table 2.6 is shown the number of datasets on which a filter method helps an algorithm perform significantly better, and the number of datasets on which the filter method helps the best performing learning algorithm perform even better. We see that for each learner there is at least one dataset on which a screening method can improve the performance. Mutual information, Relief and Fisher score have best performance among all methods. It is also clear that the screening methods can generally improve the performance of logistic regression and Naive Bayes on 4 to 5 out of the 5 datasets. When compared to best leaner without screening methods, Relief shows to be slightly weaker than the others.

In Table 2.7 is shown a "*" for each dataset and each screening method if it has a learning algorithm that obtains significantly better performance than the best learning algorithm without screening. An "=" sign shows for each dataset whether a screening method is in the same performance group as the best learning algorithm without screening. We observe that Relief only worked on the Madelon dataset. The other screening methods worked on both Gisette and Madelon datasets. It is also shown that only in a few occasions the screening methods harm the best learning algorithm. This is shown by the blank cells in the table. Overall, except Relief, the screening methods have similar performance on the five classification datasets.

In Table 2.8 is shown for each screening method-learning algorithm combination the number of datasets for which it was in the top performing group. We can observe that the screening methods

Table 2.7: Ranking of feature screening methods for classification by number of datasets where screening method was significantly better than the best performed no screening method. (larger numbers are better. * indicates appearance.)

Screening Method	Dexter	Gisette	SMK_CAN_187	Madelon	GLI_85	Total count
Chi-square Score	=	*	=	*		2
Gini Index	=	*	=	*	=	2
Relief		=	=	*	=	1
Mutual Information	=	*	=	*	=	2
T-score	=	*	=	*		2
Fisher Score	=	*	=	*	=	2
MRMR	=	*	=	*		2

Table 2.8: Number of datasets where each combination was in the top performing group.

Filter \ Learners	Boost Tree	FSA	SVM	NB	Logistic
Mutual Information	2	2	2	0	1
Gini Index	2	2	1	0	1
Chi-square Score	2	2	0	0	1
Relief	2	2	0	0	1
T-score	1	1	1	0	0
MRMR	1	1	1	0	0
Fisher Score	1	1	1	0	0
—	1	1	1	0	0

with boosted trees and FSA have the overall best performance. Among them, boosted trees and FSA with four screening methods (Chi-square Score, Gini Index, Relief and Mutual Information) have a slight advantage compared to the algorithms without screening. SVM worked well with Mutual Information. The above named four screening methods also helped Logistic regression on one dataset. Naive Bayes didn't perform well on these five datasets.

In Table 2.9, are shown the number of times each screening method was in the top performing group. In the first column, these methods were counted with respect to the learning algorithms they were applied. So there can be at most 25 counts (for each screening method there are five learning algorithms and five datasets) in each cell. The second column shows the counts with the best learning algorithm, so there can be at most 5 counts in each cell. The Mutual Information has the highest counts. It's significantly higher than no screening. Gini Index, Relief and Chi-square score also have relative higher counts when considering them together with a learning algorithm. Mutual

Table 2.9: Ranking of feature screening methods for classification by the number of times each was in the top performing group. (larger numbers are better)

Screening Method	Top performing Method-Algorithm	Top performing Method
Mutual Information	7	4
Gini Index	6	4
Chi-square Score	5	3
Relief	5	3
T-score	3	2
MRMR	3	2
Fisher Score	3	2
No Screening	3	3

Information and Gini Index have good performance on more datasets than using no screening, when considering only the best learning algorithm for each method and each dataset.

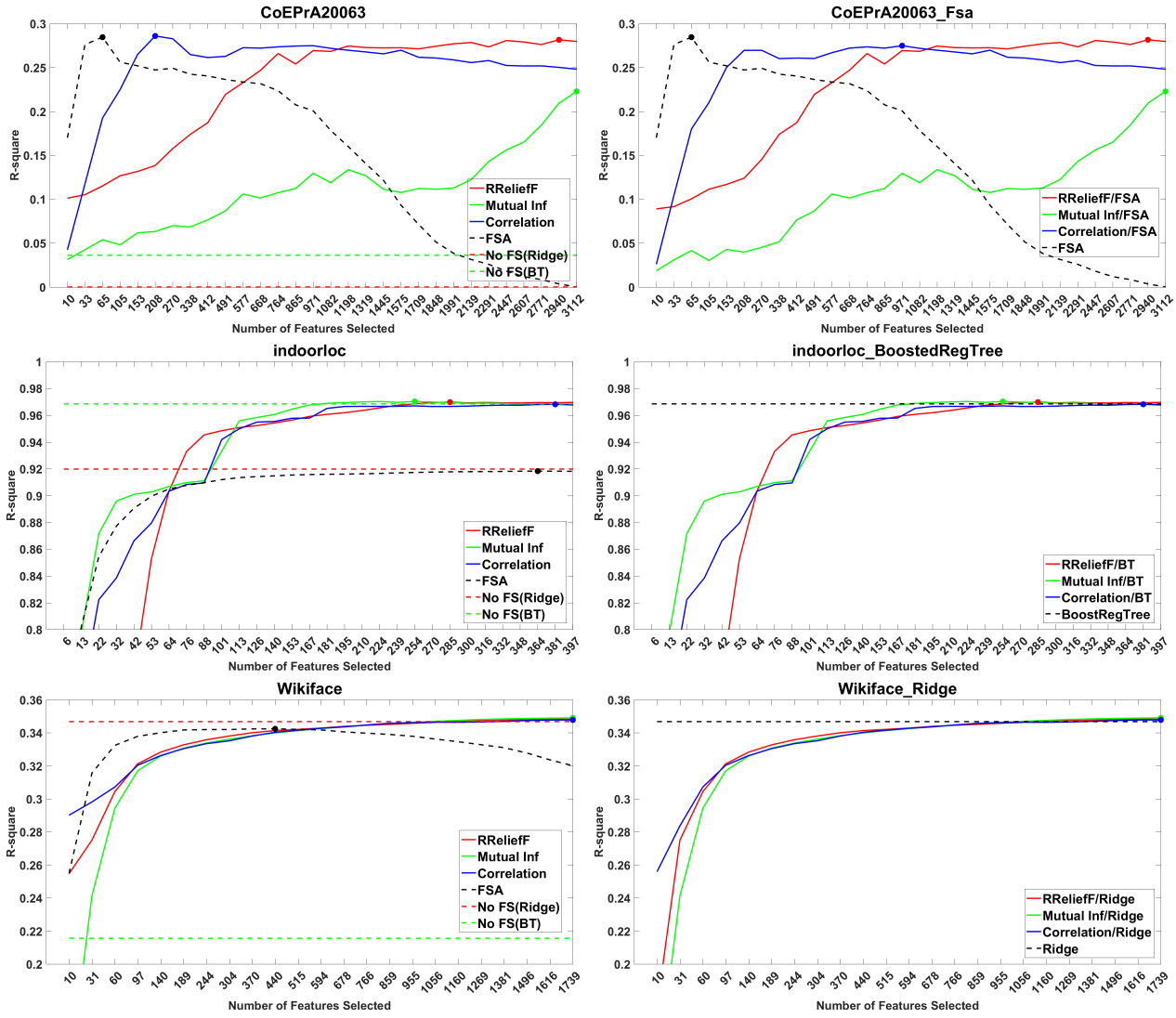

Figure 2.2: Performance plots of methods with and without feature screening. Left: for each screening method are shown the maximum R^2 value across all learners. Right: R^2 of the screening methods with the best learner for this data (ridge).

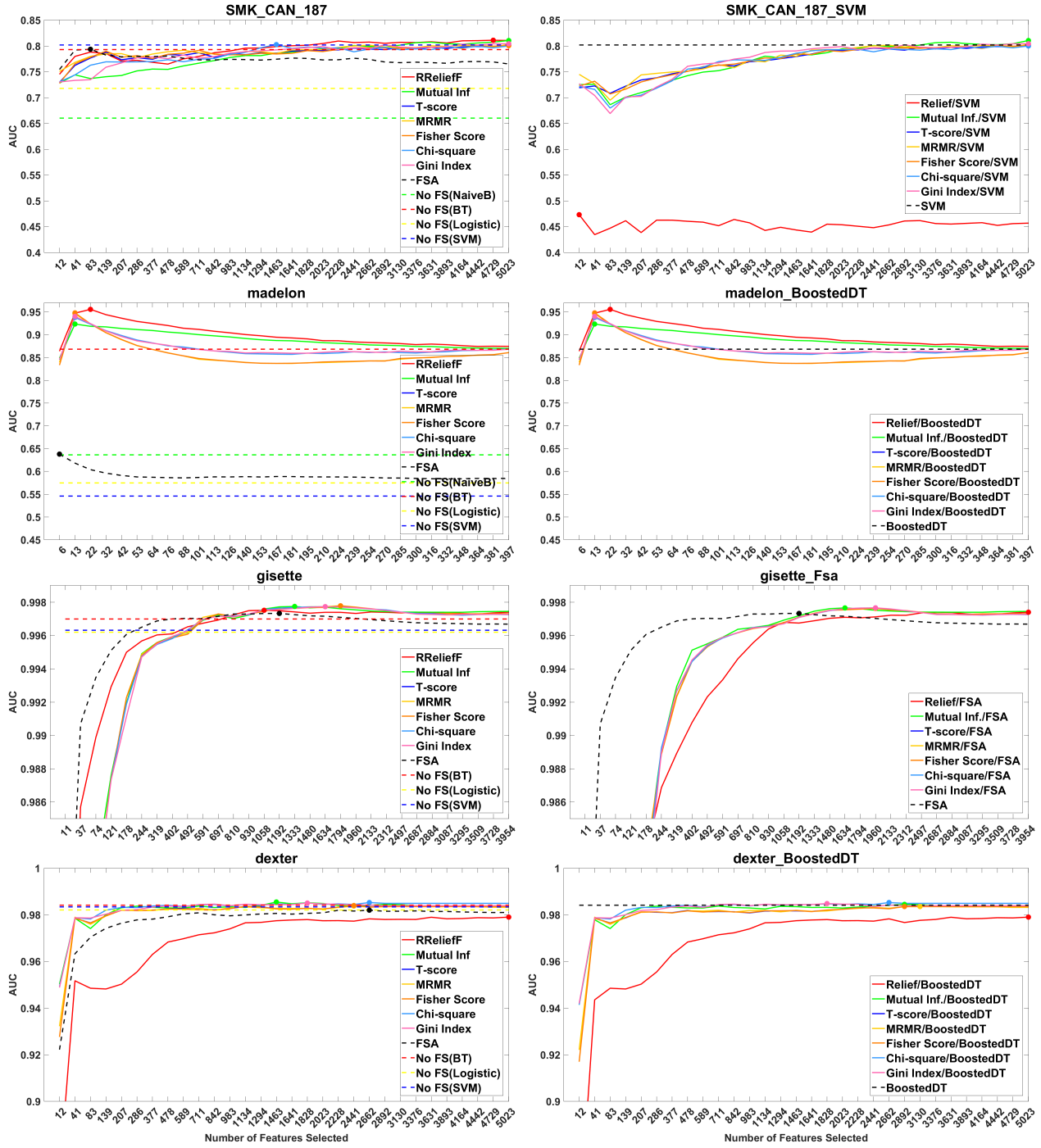

Figure 2.3: Performance plots of methods with and without feature screening. Left: for each screening method are shown the maximum R^2 value across all learners. Right: R^2 of the screening methods with the best learner for this data.

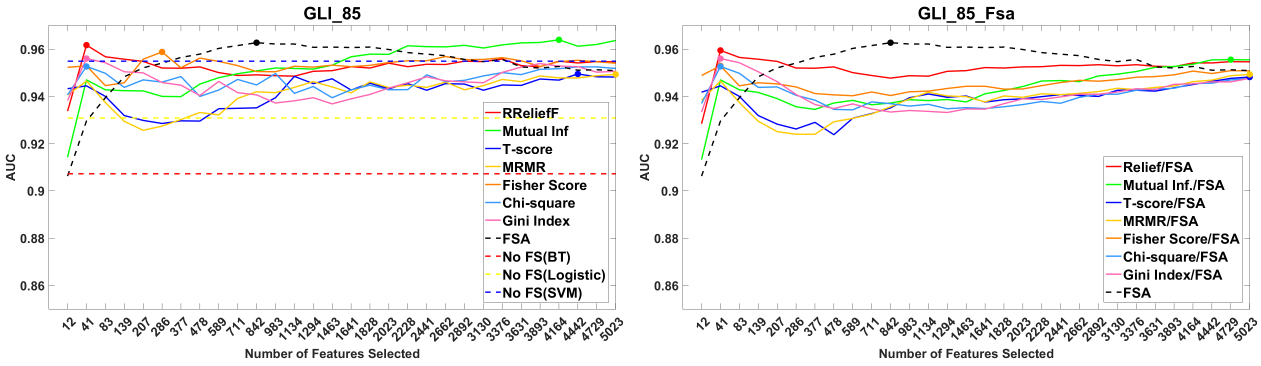

Figure 2.4: Performance plots of methods with and without feature screening. Left: for each screening method are shown the maximum R^2 value across all learners. Right: R^2 of the screening methods with the best learner for this data.

CHAPTER 3

ONLINE SCREENING METHODS

3.1 Introduction

With the explosive growth in amount of data available in recent years, the efficiency of data processing stays a heated topic in both hardware and software fields. Various approaches were emerged to tackle this challenge, such as cloud computing, batch learning, online learning, more powerful hardware and so on. These approaches mainly focus on how to increasing the loading capacity of the systems. On the other hand, the screening (filter) feature selection methods, which shows impressive performance in reducing feature dimensions of large high dimensional data and improving overall model performance, especially with ad-hoc implementations, focus on how to scale down the data before considering the loading capacity of the learning system. Screening feature selection methods are independent of the learning algorithms, which gives them a fast execution speed. They can be ideal add-ons when it comes to improving efficiency without losing performance. Built on the foundation of our previous survey on the performance of screening feature selection methods on real large datasets, we dedicate this article to introduce several novel online feature screening methods as extensions of existing offline screening methods to tackle datasets that come from a more modern environment.

In this study, five screening methods are selected from our previous survey [Wang and Barbu, 2019] to be extended to their online versions, sparse versions, and addressing model adaptation issues. Among them, T-scoreDavis and Sampson [1986] and Fisher Score [Duda et al., 2012] are mean-variance based methods, while Gini index [Gini, 1912], Chi-square score [Liu and Setiono, 1995], and Mutual Information [Lewis, 1992] are quantile-based methods. Comparative evaluations are then conducted on these methods between their online and offline versions using synthetic and real datasets. The model adaptation will also be tested against synthetic data with time-varying features. Finally we will evaluate the performance of these extended methods by showing experimental results on real datasets. Given the fact that we don't know which are the true

features in real datasets, we will show the performance of these methods by comparing the learners' predictive ability before and after applying screening methods.

3.2 Related Work

The study of feature selection on streaming data has a long history. Even before the big data boom, there already existed applications that generate and require to process streaming data under computation and memory constraints. One such field is text mining and language processing in topic recognition and spam email identification. In cooperating with the well know SpamHunting system, J.R. Méndez et al. introduced a new term selection scheme in [Méndez et al., 2007] based on Amount of Information (AI). AI is calculated from the appearance of certain terms in the text corpora. It can also be extended as a feature selection method for other types of binary categorical features. This is a method with online criteria calculation and concept adaptation capability. Concept adaptation is an essential issue online algorithms need to deal with when processing data streams over a large, possibly infinite, amount of time. The fundamental idea is that the relation between the prediction target and its features usually changes over time either gradually or drastically. The subset selected by the feature selection methods needs to be adjust to this change as well. Concept adaptation is also called concept drifting, model adaptation depending on the focus of the dynamic connection between target and features.

Feature selection methods can be divided into three categories based on their interaction with the learning algorithm: filters, wrappers and hybrids. Filters, also known as screening methods, compute a criterion for each feature and select features according to some selection scheme. They don't involve learner. The aforementioned AI method is a filter method. Wrappers wrap feature selection with any learner. They use a feature importance measure calculated by the learner to decide what features to keep. Hybrids, also known as embedded methods, rely on a sparsity inducing built-in regularizer of the learner to drop irrelevant features. One such example is the LassoTibshirani [1996].

Screening methods have the advantage of fast processing and easy integration with other system. The framework introduced in this article also fall into this category. Considering the large number of streaming high dimensional applications, we feel its advantages make it an appropriate direction to pursue.

Outside the scope of screening methods, there are other online feature selection methods in the literature. In [Carvalho and Cohen, 2006] an online learner Modified Balanced Winnow(MBW) was introduced. The authors use the absolute value between the positive weights and negative weights calculated by MBW to rank the features by their importance. This is a typical setting for the wrapper methods. Importing the idea of the trade-off between exploration and exploitation, [Wang et al., 2013] proposed a method that periodically updates feature weights, and selects feature using an arbitrary classifier.

Throughout the years, online screening methods have achieved steady development. Some of them are introduced to solely handle large data streaming without considering the drifting problem. Katakis et al. [Katakis et al., 2005] proposed a two stage mechanism for feature selection and data classification. They employed a incremental criterion calculation method. Features are selected according their rank based on the criterion such as chi-squared score and mutual information. Their incremental method is formulated for discrete features, therefore their method can not handle continuous feature. In response to the issue of concept drift for streaming data, different strategies were introduced.

One of the popular methods is sliding window. Its idea is to only consider the most recent data, making it a natural adaptation to the drifting problem. In [Masud et al., 2010] a chunk structure similar to the sliding window was used by author. Deviation weight was introduced in this article as a feature importance measurement and indicator. The feature selection is them perform based on each feature's deviation weight. However the deviation weight is only applicable to categorical features. Similarly in [Gomes et al., 2013], an contingency table is maintained for each feature from the sliding window. The tables are then used to calculate the criterion such as Mutual information and Chi-squared score and so on. The criterion mentioned in their study are only suitable for processing discrete features. They also introduced a dynamic threshold to avoid naive selection roles such as top-k features and fixed threshold.

Another popular strategy widely adapted to handle the drifting problem is the fading factor. In [Sovdat, 2014], both sliding window and fading factor are used. In the fading factor variation, feature measures such as the Gini index and entropy are calculated incrementally over the entire data stream. However the criterion they use also works with discrete data only. An interesting attempt was presented inHenke et al. [2015], which uses a month as the specific window size for

the sliding window to calculate mutual information from categorical features. This is very practical the the cases where the drift information is known.

In some studies, an estimator of criteria is used instead of exact computed criteria mentioned above. Keller et al. [Keller et al., 2015] used a k-nearest neighborhood mutual information estimator introduced by Kraskov el. [Kraskov et al., 2004] under the sliding window frame. The features are then selected based on their mutual information estimators. Instead of calculating some criterion for each feature all the way, in [Hammoodi et al., 2018] a concept drift detector detects which features drift first using Feature Velocity and Inter Quartile Range. Once a drift is detected, all features that relate to the drift will have their mutual information updated using data in a recent window. Then a ranking and selection procedure is carried among all features based on their mutual information.

Despite various situations in which the aforementioned methods can be applied, to our best knowledge, none of the existing online screening methods applies to continues features. In order to calculate an appearance based criterion such as mutual information and chi-squared score from continuous features, one needs to be able to discretize the continuous features in an online fashion. Many efforts have been put into this area. It has been proven in [Munro and Paterson, 1980] that $O(N)$ space is required for an algorithm to compute the exact quantiles in a single pass of streaming data. In order to describe a one pass data stream in a reduced space, approximate quantile calculation methods are needed. Some earlier works are [Jain and Chlamtac, 1985] and [Agrawal and Swami, 1995]. Their algorithms are used to calculate uniform quantiles in a single pass. In [Gama and Pinto, 2006] a two-stage framework was proposed to obtain discrete binned data from continuous data. The suggested method first constructs many equal width intervals to capture and update the partition of incoming sample points. At query time, it aggregates the collected intervals to generate equal-width histograms or equal-frequency histograms. As intuitive as it is, this is a non-deterministic method, in the sense that there are no deterministic guarantees on the estimation error.

Manku et al. introduced a single pass algorithm in [Manku et al., 1998] to compute a deterministic ϵ-approximate uniform quantile summary. It requires prior knowledge of the sample size N and has a space complexity of $O(\frac{1}{\epsilon}\log^2 \epsilon N)$. Another algorithm that does not require prior knowledge of N was also proposed by Manku et al. in [Manku et al., 1999]. The space complexity for

this algorithm is $O(\frac{1}{\epsilon}(\log^2 \frac{1}{\epsilon} + \log^2 \log \frac{1}{\delta}))$ with a failure probability of δ. A more recent approach (the GK algorithm) to compute a deterministic ϵ-approximate quantile summary on a single pass of streaming data without the prior knowledge of N was introduced by Greenwald et al. in [Greenwald et al., 2001]. This method imposed a tree structure. It is an improvement of Manku's algorithm with a space bound of $O(\frac{1}{\epsilon} \log \epsilon N)$. In [Zhang and Wang, 2007], an improvement was made on the GK algorithm to significantly reduce the computational cost. The computational cost of this multi-level quantile summary algorithm is $O(N \log \frac{1}{\epsilon} \log \epsilon N)$. It is shown in their experiments that it can achieve about 200 - 300 $\times$ speedup over the GK algorithm. Its storage requirement of $O(\frac{1}{\epsilon} \log^2 \epsilon N)$ is higher than the GK algorithm.

Recent work in XgboostChen and Guestrin [2016] extended [Greenwald et al., 2001] and [Zhang and Wang, 2007] with a focus on processing weighted data. This saves space and improves the summary accuracy when dealing data streams containing duplicate values. The algorithm introduced in their article has the same guarantee as the GK algorithm. It can be plugged into all GK frameworks and its extensions.

There also exists works that focus on other aspects of calculating quantile summaries. In [Lin et al., 2004] was introduced an algorithms to compute uniform quantiles over sliding windows. Cormode et al. [Cormode et al., 2006] proposed an algorithm to handle the biased quantile problem. It has a storage bound of $O(\frac{\log U}{\epsilon} \log \epsilon N)$ and time complexity of $O(\log \log U)$ where N is the sample stream size and U is the size of the domain from which the sample points are drawn. Efforts were also made to compute approximate quantiles for distributed streams and sensor networks. Greenwald et al. [Greenwald and Khanna, 2004] proposed an algorithm for calculating ϵ-approximate quantiles for sensor network applications. In [Shrivastava et al., 2004] an algorithm with space complexity of $O(\frac{1}{\epsilon} \log U)$ was proposed to compute medians and other quantiles in sensor networks.

In this study, we implemented mean-variance based feature screening methods using moving averages. We will also introduce quantile based methods based on the weighted quantile summary. We extended work in [Chen and Guestrin, 2016] to generate accurate live bin counts on demand. Our proposed algorithm also integrates ways to handle sparse streaming data as well as streaming data featuring concept drift, as an adaptation to the needs of modern applications.

3.3 Methods

The online screening methods we studied are divided into two categories based on their fundamental principles. The T-score [Davis and Sampson, 1986] and Fisher score [Duda et al., 2012] are based on moving averages. Mutual Information [Lewis, 1992], Chi-square score [Liu and Setiono, 1995] and Gini index [Gini, 1912] are based on quantile summaries.

3.3.1 Mean-Variance Based Methods

T-score. For a feature x_j, its T-score is calculated as:

$$T_j = \frac{|\mu_1 - \mu_2|}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}} \tag{3.1}$$

where μ_c, n_c, and σ_c denote the mean, sample count, and standard deviation of the values of observations belonging to class c. The higher the score, the more relevant the feature is to the target variable.

Fisher Score. Similarly, the Fisher score of feature x_j is defined as:

$$Fisher_j = \frac{\sum_{c=1}^{C} n_c(\mu_c - \mu)^2}{\sum_{c=1}^{C} n_c \sigma_c^2} \tag{3.2}$$

where μ is the mean of the feature, and μ_c, n_c, and σ_c have been defined above.

It is clear that equation (3.1) and (3.2) are calculated from basic components such as means and variances. To generalize them to a streaming data setting is quite straightforward.

Without losing generality, we assume that a sample arrives at each time step $t = 1, 2, 3, ..., n$. At time n, the running average μ_{nj} and running mean of squared MS_{nj} of the j-th feature that form the sufficient statistics are:

$$\begin{aligned}
\mu_{1j} &= x_{1j} \\
\mu_{2j} &= \frac{\mu_{1j}}{2} + \frac{x_{2j}}{2} \\
&\vdots \\
\mu_{nj} &= \frac{n-1}{n}\mu_{(n-1)j} + \frac{1}{n}x_{nj}
\end{aligned}$$

And:

$$
\begin{aligned}
MS_{1j} &= x_{1j}^2 \\
MS_{2j} &= \frac{MS_{1j}}{2} + \frac{x_{1j}^2}{2} \\
&\vdots \\
MS_{nj} &= \frac{n-1}{n} MS_{(n-1)j} + \frac{1}{n} x_{nj}^2
\end{aligned}
$$

Consequently the variance of j-th feature at time n can be written as:

$$\sigma_{nj}^2 = MS_{nj} - \mu_{nj}^2 \tag{3.3}$$

Therefore T-score and Fisher Score can incrementally maintain exact calculation as new samples arrive.

In the scenario of sparse input, since the zero values are no-shown in data stream, one only need to accumulate the running average and running mean of squared with showed value and keep the record of sample count.

In order to adapt concept drifting, a fading factor strategy is used as a penalty on the history incremented statistics:

$$\mu_{nj} = \alpha * \mu_{(n-1)j} + x_{nj}$$

$$MS_{nj} = \alpha * MS_{(n-1)j} + x_{nj}^2$$

Where α is a fading factor that takes a user set value in $(0, 1)$. When adaptation is required with sparse input, a time anchor is employed for each feature in every class to recording the last appearance of non-zero values n_{last}. It is equal to current sample count. Accumulated statistics then updated the next time a non-zero value appears:

$$\mu_{update,j} = \alpha^{(n-n_{last}-1)} * \mu_{last,j}$$

$$\mu_{nj} = \alpha * \mu_{update,j} + x_{nj}$$

$$MS_{update,j} = \alpha^{(n-n_{last}-1)} * MS_{last,j}$$

$$MS_{nj} = \alpha * MS_{update,j} + x_{nj}^2$$

Where $\mu_{last,j}$ and $MS_{last,j}$ is the penalized running average and running mean of squared at last appearance of a non-zero value.

3.3.2 Bin Count Based Methods

Criteria. Mutual Information. For a feature x_j, its mutual information can be calculated as:

$$I(X_j, \mathbf{y}) = \sum_{b=1}^{B} \sum_{c=1}^{C} P(x_j \in bin_b, \mathbf{y} = c) \log \frac{P(x_j \in bin_b, \mathbf{y} = c)}{P(x_j \in bin_b) P(\mathbf{y} = c)} \tag{3.4}$$

where $P(x_j \in bin_b, \mathbf{y} = c)$ is the joint probability of having feature values fall into bin_b and label value equal to c. $P(x_j \in bin_b)$ and $P(\mathbf{y} = c)$ are the marginal probabilities.

In the case of samples with discrete feature values, the probability can be expressed as:

$$P(x_j \in bin_b, \mathbf{y} = c) = \frac{n_{x_j \in bin_b, \mathbf{y}=c}}{n}$$

$$P(x_j \in bin_b) = \frac{n_{x_j \in bin_b}}{n}$$

$$P(\mathbf{y} = c) = \frac{n_{\mathbf{y}=c}}{n}$$

where n, $n_{x_j \in bin_b, \mathbf{y}=c}$, $n_{x_j \in bin_b}$, $n_{\mathbf{y}=c}$ denote the sample count that fall into respective value groups.

Chi-squared Score. With a similar definition of n's, the chi-squared score of a feature x_j can be defined as:

$$\chi_j^2 = \sum_{b=1}^{B} \sum_{c=1}^{C} \frac{(n_{x_j \in bin_b, \mathbf{y}=c} - \hat{n}_{x_j \in bin_b, \mathbf{y}=c})^2}{\hat{n}_{x_j \in bin_b, \mathbf{y}=c}} \tag{3.5}$$

where:

$$\hat{n}_{x_j \in bin_b, \mathbf{y}=c} = \frac{n_{x_j \in bin_b} n_{\mathbf{y}=c}}{n}$$

Gini Index. For a given feature x_j, let $A_h = \{i, x_{ij} \leqslant h\}$ denote the number of samples whose values of the j-th feature is smaller than or equal to h and $B_h = \{i, x_{ij} > h\}$. Its Gini Index can be expressed as:

$$Gini_j = P(A_h)(1 - \sum_{c=1}^{C} P(C_c|A_h)^2) + P(B_h)(1 - \sum_{c=1}^{C} P(C_c|B_h)^2) \tag{3.6}$$

where $P(A_h)$ is the number of samples in subset A_h divided by the number of total samples. $P(C_c|A_h)$ is the conditional probability of samples having label c given that they are in subset A_h. Let $n_{x_j \in A_h, \mathbf{y}=c}$ denote the number of samples in A_h with label c. Let $n_{x_j \in A_h}$ denote the number of samples in A_h. Then $P(C_c|A_h)$ can be calculated as $n_{x_j \in A_h, \mathbf{y}=c}/n_{x_j \in A_h}$. The same goes for $P(B_h)$ and $P(C_c|B_h)$. h is chosen to give the minimum Gini Index for each feature.

In order to compute aforementioned criterion from data stream with continuous values. A proper online discretization method must be applied. Therefore based on the ϵ-approximate quantile summary structure in [Greenwald et al., 2001], [Zhang and Wang, 2007] [Chen and Guestrin, 2016], we introduce an improved implementation algorithm to generate an on demand bin count from data stream with continuous values. Furthermore, we extend our algorithm to adapt to sparse input and concept drifting scenario. In the following segment we will first present an overview of ϵ-approximate quantile summary structure and its operation. Then we will illustrate our extended methods in detail.

Quantile Summary. The basic idea is to use several container like sub-summaries s to store approximated ranking information of partial data stream. In turn, an aggregated summary S of sub-summaries can describe the entire data stream. Various operation are conducted periodically to maintain the estimations in these containers such that at any time n, the summary $S(n)$ can answer any r-quantile query with ϵn precision.

A sub-summary consists of several tuples $s = \{T_1, T_2, ..., T_b\}$. Each tuple in the form of T=$(v, \tilde{r}^-, \tilde{r}^+, \tilde{w})$ describes a number of similar data points from the data stream. v denotes the data value estimation this tuple represents. $\tilde{r}^-$ and $\tilde{r}^+$ are the lowest and highest estimated ranks of this value in the current sub-summary. $\tilde{w}$ is the accumulated weight value which represents how many data points this tuple covers. The sub-summary s is sorted by the values v. Given a small input stream $Q = \{(x_1, w_1), (x_2, w_2), ..., (x_n, w_n)\}$, where (x_i, w_i) is a data point. x_i denotes its value and w_i is its weight. Usually data weights are set to 1. For each tuples, two rank functions and a weight function are defined.

$$r^-(v) = \sum_{(x,w) \in Q, x < v} w \tag{3.7}$$

$$r^+(v) = \sum_{(x,w)\in Q, x\leqslant v} w \tag{3.8}$$

$$w(v) = r^+(v) - r^-(v) = \sum_{(x,w)\in Q, x=v} w \tag{3.9}$$

The weight of the entire sub-summary is defined as:

$$w(s) = w(Q) = \sum_{(x,w)\in Q} w \tag{3.10}$$

Without loss of generality, in the rest of this article, weight will be used to refer to the calculated weight value of $w(v)$. Rank will be used to refer to the calculated rank value of $r^+(v)$ or $r^-(v)$.

Summary S is consist of a multi-level sub-summary structure, shown in Figure 3.1, when it is not queried. It is used to maintain the desired precision as well as speed up the calculation. L is the total number of levels. s_l denotes the sub-summary at level $l, l = 0, 1, ..., L$. The whole data stream is divided into consecutive segments of size $b = \left\lceil \frac{L}{\epsilon} \right\rceil$, where L is the largest integer that makes $b2^{(L-1)} \leqslant N$.

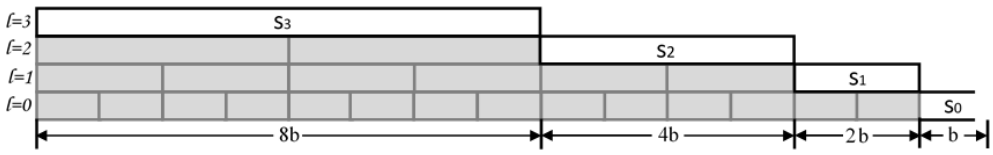

Figure 3.1: Multi-level summary: The length of s_l in the figure represents its coverage of data points. s_0 contains the summary for the most recent data input block. s_l consists the summary of the oldest 2^l data blocks. At each level, s_l is maintained as an ϵ_l-summary.

At the lowest level, s_0 is defined to hold all recently arrived points until it researches size b. The tuples in s_0 are constructed with

$$v = x_i, \tilde{r}^-(x_i) = r^-(x_i), \tilde{r}^-(x_i) = r^-(x_i), \tilde{w}(x_i) = w(x_i) \tag{3.11}$$

Therefore s_0 is a 0-approximate summary. It can answer all query questions exactly. Algorithm 2 shows the basic procedures of summarizing when a new data point x_i in the data stream arrives. In the algorithm, s_{temp} denotes a temporary sub-summary. s_l denotes the lth level in the multi-level summary structure mentioned above.

PRUNE$(s, \frac{k}{2})$ is an operation that converts a sub-summary s with size k and precision ϵ_0 into a sub-summary with size $\frac{k}{2} + 1$ and precision $\epsilon_0 + \frac{1}{k}$. It is shown in [Zhang and Wang, 2007] that each level in the summary can maintain an error less than ϵ.

Algorithm 2 General Procedure for Quantile Summary

Input: x_i, where i=1, ..., t

1: push x_i into s_0
2: **if** $size(s_0) < b$ **then**
3: go back to line 1
4: **else**
5: s_{temp}=PRUNE(s_0, $\frac{size(s_0)}{2}$)
6: clear s_0
7: **for** $l = 1, ..., L$ **do**
8: **if** $size(s_l)$ is 0 **then**
9: $s_l = s_{temp}$
10: clear s_{temp}; break
11: **else**
12: s_{temp}=MERGE(s_{temp}, s_l)
13: s_{temp}=PRUNE(s_{temp},$\frac{size(s_{temp})}{2}$)
14: **if** $size(s_{temp}) < b$ **then**
15: $s_l = s_{temp}$; clear s_{temp}; break
16: **else**
17: clear s_l
18: **end if**
19: **end if**
20: **end for**
21: **end if**
22: **Output:** S=MERGE($s_0, s_1, ..., s_L$)

In the PRUNE operation, maximum g tuples are chosen from the input sub-summary according to position indicator $d = \frac{i-1}{g}w(s), i = 1, 2, ..., b+1$. Algorithm 3 shows how to use the query operation $Q(s, d)$ to choose tuples to form the new sub-summary. $\tilde{r}^-(x_i), \tilde{r}^-(x_i), \tilde{w}(x_i)$ of the selected tuples are copied from the original sub-summary.

Another operation is MERGE. MERGE(s_1,s_2) combines two sub-summaries into one sub-summary. Tuples from two sub-summaries s_1 and s_2 are sorted together by their value v. For each unique value v from s_1 and s_2, $\tilde{r}^-(x_i), \tilde{r}^-(x_i), \tilde{w}(x_i)$ are updated as follows

$$\tilde{r}^-(v) = \tilde{r_{s_1}}^-(v) + \tilde{r_{s_2}}^-(v) \tag{3.12}$$

$$\tilde{r}^+(v) = \tilde{r_{s_1}}^+(v) + \tilde{r_{s_2}}^+(v) \tag{3.13}$$

Algorithm 3 Query Function Q(s,d)

Input: d: $0 \leqslant d \leqslant w(s)$, s is a sub-summary with tuple value $v_i = x_i$, $i = 1, 2, ..., k$

if $d < \frac{1}{2}[\tilde{r_s}^-(x_1) + \tilde{r_s}^+(x_1)]$ **then**
  return x_1
end if
if $d \geqslant \frac{1}{2}[\tilde{r_s}^-(x_k) + \tilde{r_s}^+(x_k)]$ **then**
  return x_k
end if
Find i such that
$\frac{1}{2}[\tilde{r_s}^-(x_i) + \tilde{r_s}^+(x_i)] \leqslant d < \frac{1}{2}[\tilde{r_s}^-(x_{i+1}) + \tilde{r_s}^+(x_{i+1})]$
if $2d < \tilde{r_s}^-(x_i) + \tilde{w_s}(x_i) + \tilde{r_s}^+(x_{i+1}) - \tilde{w_s}(x_{i+1})]$ **then**
  return x_i
else
  return x_{i+1}
end if

$$\tilde{w}(v) = \tilde{w_{s_1}}(v) + \tilde{w_{s_2}}(v) \tag{3.14}$$

Let the precisions of two sub-summaries before merging to be ϵ_a and ϵ_b. The precision of merged sub-summary s_{temp} is $\max(\epsilon_a, \epsilon_b)$ [Chen and Guestrin, 2016]. MERGE operation can also be apply to more than two sub-summaries. It is shown, at the end of Algorithm 2, the aggregated summary S is the out come of MERGE operation over all sub-summaries. Therefore, S in its result form has the same structure as s.

It is shown in [Zhang and Wang, 2007] that the outcome summary S is an ϵ-approximate summary of the entire stream.

According to Algorithm 2, whether to perform a PRUNE or MERGE operation is based on the segment size b which is determined by the stream size N. In order to obtain a summary from a data stream with stream size N unknown, the input data stream is divided into pieces of disjoint sub-streams $B_i, i = 0, 1, ..., m$. B_i has size of $\frac{2^i}{\epsilon}$ and covers data arriving in the time interval $[\frac{2^i-1}{\epsilon}, \frac{2^{i+1}-1}{\epsilon})$. With fixed sub-stream size aggregated summaries can be obtained. Due to the fact that the output of S and s have the same structure, S can too perform PRUNE and MERGE operation with other S. Therefore after obtaining S_i for each sub-stream B_i, a multi-level summary structure $[S]$ now can be constructed from the summary of each sub-stream S_i. The procedure is illustrated below.

1. The summary S_c of current sub-stream B_c is updated and maintained until the last data point in B_c has arrived. $\epsilon' = \frac{\epsilon}{2}$ is used to set size limit.

2. A $\frac{\epsilon}{2}$-approximate summary is obtained as an output of S_c. The output is then set to PRUNE with the desired size of $\frac{2}{\epsilon}$ and assigned to S_i.

3. A set of summaries of all sub-streams $\widetilde{S} = \{S_0, S_1, ..., S_m\}$ is computed. $[S]$ is obtained by MERGE summaries in $\widetilde{S}$.

Exact Weight Update. When used as a query algorithm, only the rank r and value v of the tuple will be used to answer the question. According to the original PRUNE operation in section 3.3.2, the selected tuples along with their stored elements are directly moved to the resulting summary. Although this behavior causes the lost of half of the elements that store weight values. It still guarantees the ϵ maximum error as shown in [Chen and Guestrin, 2016] [Greenwald et al., 2001] [Zhang and Wang, 2007]. However, to the end of providing accurate bin count as inputs to these bin count based criteria, each tuple is treated as a mini-bin. The weight w of each tuple will be used to calculate the final bin count. Therefore, the preservation of the complete weight values is required.

In the PRUNE operation, after each selection, instead of carrying elements directly from original tuples to tuples in new summary, different procedures are taken. Let the value of each tuple associate with original summary be v_k, $k = 1, 2, 3, ..., i, ..., j, ..., b$. i is the index for last selected tuple. j is the index of currently selected tuple. Let the value of each tuple associate with output summary be u_h, $h = 1, 2, 3, ..., q, ..., b/2$. u_q is the corresponding tuple derived from v_j. Following adjustments are made:

$$
\begin{aligned}
u_q &= v_j \\
\tilde{r}^-(u_q) &= \tilde{r}^-(v_{i+1}) \\
\tilde{r}^+(u_q) &= \tilde{r}^+(v_j) \\
\tilde{w}(u_q) &= \sum_{k=i+1}^{j} \tilde{w}(v_k)
\end{aligned}
$$

The definition of ϵ-approximate quantile summary states that let Q be the set of all data points covered by the summary, if for any $x \in Q$

$$\tilde{r}^+(x) - \tilde{r}^-(x) - \tilde{w}^+(x) \leqslant \epsilon w(Q) \tag{3.15}$$

The above adjustment leads the left hand side of the inequality to zero which satisfied the requirement of a ϵ-approximate quantile summary.

Quantile Binning. With the weight of each data point set to 1, the weight element in a tuple can represent the number of data points this tuple covered. Using procedure in last two sections, a final summary $[S]$ consists of m tuples is generated. $[S]$ is then further aggregated into smaller number of denser bins. For the purpose of comparability with our offline method survey [Wang and Barbu, 2019], procedures that mimic the discretization in [Nguyen, 2014] are introduced. Details are shown in Algorithm 4. d_{inter} denotes the interval length when data points are equally divided into K segments. Bin_k denotes the bin count in kth Bin. p_i is the position index of ith cutoff point. $T_j.w$ represents the weight of the jth tuple.

Algorithm 4 Bin Aggregation Procedure

Input: $[S] = T_1, T_2, ..., T_m$, N: number of total data so far, K: user defined number of final bins.

$d_{inter} = \lfloor \frac{N}{K} \rfloor$, h = 0
for $i = 1, ..., K-1$ **do**
  $p_i = i * d_{inter}$
end for
for $k = 1, ..., K$ **do**
  $Bin_k = 0$
end for
for $j = 1, ..., m$ **do**
  $Bin_h = Bin_h + T_j.w$
  if $|Bin_h| \geqslant p_h$ **then**
    $h_{temp} = h$
    h = h + 1
    while $|Bin_{h_{temp}}| \geqslant p_h$ **do**
      h = h + 1
    end while
  else
    continue
  end if
end for
Output: $Bin_k, k = 1, 2, ..., K$

Criteria score in section3.3.2 can then be calculated using Bin_k

Sparse Input. Different from using running sum for mean-variance based methods, the zero values are assigned weight one. Therefore in the scenario of sparse input, zero values need to be processed through the summary. Processing zero values every time one shows up will be computationally inefficient. We take the advantage of the fact that sparse data has enormous zero values and quantile summary directly stacks the weights of identical data values together. The total number of zero values are recorded. When the algorithm is called to provide feature importance score, a single data point (x, w), with x equals to 0 and w equals to the recorded number, is pushed through the algorithm before aggregating the summary.

Model adaptation. In the situation of concept drifting, a fading factor strategy is still adopted. Let α denote the fading factor, w_i be the weight value of data point at i time and W_i be the weight values in the multi-level quantile summary at i time.

$$W_i = \alpha W_{i-1} + w_i \tag{3.16}$$

Notice in equation 3.16, the weight in W_{i-1} needs to be updated every time a new data point arrive. This includes all the tuples on all the levels in the summary. Such high frequency repeatedly calculation is very time consuming. Therefore we split update process into two parts. The first part, we conduct update only on most recently established tuples in s_0(see Fig 3.1 to refresh memory of s_0). The second part, based on the number of data points covered in s_0, we only penalize the weights in multi-level structure whenever a PRUNE and MERGE operation initialized by s_0 happens. Let the number of data points in s_0 be k and the weights in multi-level summary be W_i. The update for part two become:

$$W_i = \alpha^k W_{i-1} \tag{3.17}$$

Sparse Inputs When sparse inputs are received in a model adaptation setting, since the weight of each data point is penalized according to the order that it arrives, injecting the weights of all zero value data points at the end can not provide the correct penalized weight. In order to keep weight accumulation matching with data point order for the zero value data points, time anchors and universal weight maps are employed. The universal weight map is a vector that store the accumulated penalized weight at each timestamp for each class. These maps are used across all features. The time anchors are designed to record the timestamp of the last non-zero data point.

When incoming data stream has greater or equal to two class. Simply recording the class label and recover the penalized weight for zero values by repeatedly penalizing the weights in summary so far is extremely time consuming. On the contrary, universal weight map and time anchor can achieve the same purpose with very small extra storage space and faster computing speed without trigger inter-class penalization complex. Given universal weight map recorded up until now for each class. The weight for all zero values between last non-zero data point and current non-zero data point can be calculated. Let time indices for last non-zero data point be a and current non-zero data point be b.

$$\mathbf{w}_c = M_{b,c} - M_{a,c} * \alpha^{(b-a)} \tag{3.18}$$

$\mathbf{w}_c$ denotes the recovered weight for zero values. $M_{b,c}$ indicates the recorded weight value for c class at time index b. When a non-zero value arrives, $\mathbf{w}_c$ is calculated and added to summary before any other procedures.

3.3.3 Minibatch

It is noticed during our experiments that for non-spare inputs, the summaries of all features are required to be visited. This behavior builds increased computing time. Therefore within a reasonable storage budget, batch data handling are integrated with the algorithm, as it gives a considerably acceleration to the algorithm by reducing the visit frequency. This batch procedure is denoted by minibatch in the rest of the text.

3.4 Evaluation of Online Screening Methods

3.4.1 Online-Offline Methods Comparison

To perform the experiments, each dataset is processed in a one-pass fashion by both online and offline version of screening methods respectively. Each dataset is passed through each algorithm and parameter setting once and the weight scores for all features are calculated. For quantile summary based methods, we use $K = 5$ quantile bins throughout all our experiments. We test the computation time by fixing either the minibatch size or the precision parameter ϵ and varying the other parameter. When testing the approximation accuracy, we only fix the minibatch size. The fixed values are 250 for the minibatch and 0.001 for ϵ. The varying ranges are $\epsilon = \frac{1}{f}$, where $f \in \{5, 50, 100, 500, 1000, 1500, 2000\}$ and minibatch$= 2^k$, where $k \in \{0, 1, 2, ..., 11\}$. The feature

rank is calculated by sorting the feature weights monotonically according to importance score obtained by the corresponding screening method. Features that have low rank value are more important (i.e. feature with rank value 1 is the most important). For Gini index, the feature that has smaller weight has lower (better) rank value. On the contrary, for the other methods, the feature that has larger weight has lower rank value. Feature weights and feature ranks were used to construct different kinds of tables to evaluate the performance of online methods compared to offline methods.

Construction of Comparison Tables. Five types of tables are constructed according to minibatch size, feature weights and feature rank.

1) *The influence of minibatch size on computation time.* This table shows the computation time (in milliseconds) of the online quantile compared to the offline quantile when varying the minibatch size. A minibatch size of one is equivalent to an online method without minibatch processing.

2) *The influence of the precision parameter ϵ on computation time.* This table shows the computation time (in milliseconds) of the online quantile compared to the offline quantile when varying the value of ϵ.

3) *The influence of ϵ on count differences per feature per bin.* This table shows the average count differences between online quantile and offline quantile when varying the value of ϵ. Each cell gives the average count difference per feature per bin.

4) *The influence of ϵ on score accuracy.* This table shows the mean score difference ratio between online methods and offline methods when varying the value of ϵ. The mean score difference ratio is calculated by

$$\text{DR} = \frac{1}{p}\sum_{j=1}^{p}\frac{|w_{on}^{j} - w_{off}^{j}|}{\max(W_{off}) - \min(W_{off})}, \tag{3.19}$$

where p is the total number of features. W_{on} and W_{off} are the score vectors of all features generated from online and offline methods. w_{off}^{j} and w_{off}^{j} are the scores of the j-th feature from online and offline methods.

5) *The influence of ϵ on the rank accuracy among top 10% most important features.* This table shows the average unmatched ranking ratio of online methods with respect to corresponding offline methods when varying the value of ϵ. Only the top 10% of features ranked by offline scores are involved. The average mis-rank ratio is calculated as $\frac{\sum_{j=1}^{p} r_{on}^{j} \neq r_{off}^{j}}{p}$, where r_{off}^{j} and r_{off}^{j} are the rank values of the j-th feature from online and offline methods.

Table 3.1: The datasets used for evaluating the online/offline screening methods.

Dataset	Learning type	Feature type	Number of features	Number of observations
Dorothea [Guyon et al., 2005]	Classification	Continuous	100,000	1,150
Url [Ma et al., 2009]	Classification	Continuous	74,110	16,000
Kdd12 [Juan et al., 2016]	Classification	Continuous	48,957	16,000

Data sets. Apart from the classification datasets from Table 2.1, three more datasets were used. Specific dataset details are given in Table 3.1.

Dorothea is part of the NIPS 2003 Feature selection challenge and is also available on the UCI Machine Learning Repository. We combined the training set and validation set in order to get a larger sample body. The Url dataset contains a total of 121 data files, one for each monitored day. We only used data from Day0 in our experiments. The Kdd12 dataset originates from the second track of the KDD Cup 2012. The raw version can be found on kaggle.com, made available by the organizers and Tencent Inc. The data we use comes from LIBSVM [Chang and Lin, 2011]: a library for support vector machines. Only the first 16000 samples were used in our experiments.

Results. The following results are based on the output generated using Matlab 2018b [Mat, 2018]. For the offline screening methods we used the same Matlab 2018b implementations as those in Chapter 2. The online screening methods were implemented by ourselves.

Comparison of the Moving Average Based Methods. It is shown in Section 3.3.1 that the moving average based online screening methods can achieve exactly the same result as their offline version. Therefore we don't provide any real data analysis here.

Comparison of the Online Quantile Based Methods. First we conducted a study of the computation time of the online quantile based methods. Here we only evaluate the time from when the data was input data to when the sample points counts in each bin were returned. We first conducted an experiment to evaluate the relation between computation time and minibatch size to estimate the minibatch size that we were going to use to obtain an acceptable speed in our following experiments. In this experiment, the ϵ was fixed at 0.001. The minibatch size was varied as 2^k, where $k = 0, 1, 2, ..., 11$. The result is shown in Table 3.2, where the time is measured in milliseconds. Compared across all datasets, it can be seen that the computation times are gradually stable for

Table 3.2: Influence of minibatch size on computation time for online quantile compared to offline quantile.

minibatch	url	SMK_CAN_187	dexter	dorothea	gisette	kdd12	madelon
offline	135,878	538	872	12,927	4,260	93,163	181
1	303,490	1,198	969	27,917	3,448	191,194	109
2	220,596	811	756	20,562	2,509	124,219	86
4	150,464	749	542	14,560	2,269	75,610	83
8	97,157	671	358	11,829	1,968	52,788	84
16	67,665	583	248	10,590	1,472	40,366	68
32	51,140	628	217	10,458	1,367	32,890	71
64	43,186	605	194	9,569	1,520	27,870	68
128	37,937	572	182	9,875	1,329	25,511	66
256	35,712	589	174	11,146	1,296	24,427	65
512	36,117	578	157	10,628	1,421	22,987	64
1024	36,061	586	160	10,970	1,251	22,694	73
2048	35,636	566	197	11,060	1,245	22,093	76

minibatch sizes lager than 128, with an acceptable fluctuation. For most datsets, the computation speed of the online quantile outperforms that of the offline quantile long before the minibatch size reaches 128. SMK_CAN_187 has only 187 observations. There the minibatch doesn't improve computation time after minibatch size 128. As a result 250 was selected as minibatch size in all of the following studies.

Table 3.3: Influence of ϵ on computation time for online quantile compared to offline quantile.

epsilon	url	SMK_CAN_187	dexter	dorothea	gisette	kdd12	madelon
offline	130,177	517	735	11,961	4,018	87,891	185
0.2	15,451	489	144	1,797	935	8,443	92
0.02	13,117	374	151	1,847	889	8,433	72
0.01	13,490	319	127	1,925	860	8,581	66
0.002	16,314	481	177	2,621	913	10,557	59
0.001	19,094	758	280	4,285	1,176	12,371	65
0.00066667	487,828	1,303	337	5,461	1,024	250,470	64
0.0005	934,553	1,458	565	6,409	1,439	235,280	60

The next experiment was conducted to verify that in a real data scenario, computation time increases as ϵ decreases. In Table 3.3 is shown that it is certainly the relation between computation speed and ϵ. Furthermore, in some case where the datasets have large sample size e.g. url and kdd12, a low ϵ value can cause the online methods to use more computation time than offline

Table 3.4: Influence of ϵ on count differences between online and offline quantiles (results are reported per feature per bin).

epsilon	url	SMK_CAN_187	dexter	dorothea	gisette	kdd12	madelon
0.2	0.086	4.06	0.012	0	8.21	0	76.09
0.02	0.0079	0	0.0011	0	0.7	0	7.57
0.01	0.00045	0	0.00047	0	0.17	0	1.18
0.002	3.80E-05	0	0	0	0	0	0
0.001	1.60E-05	0	0	0	0	0	0
0.00066667	0	0	0	0	0	0	0
0.0005	0	0	0	0	0	0	0

methods. To improve the computation speed, larger minibatch size ought to be used. Here in order to maintain the consistency, we still use 250 as minibatch size.

The following experiments are regarding accuracy. In Table 3.4 is shown the count differences between online and offline quantile methods when the ϵ value was varied. The results are reported as sample count differences per bin per feature. Although different datasets have different schedule, all differences were annihilated after ϵ decreases below 0.001. The online quantile method can in practice achieve the same counts per bin as the offline quantile when ϵ is lower than a certain level. Cross checking with Table 3.3 and Table 3.2, we can see that online methods can still achieve speed advantage with little to no error in bin count estimation.

Table 3.5 shows the accuracy measure from Eq. (3.19) of the scores calculated by three online methods compared to their offline counterparts when the ϵ value was varied. The scores were computed using the Chi-square score, Gini index and Mutual information respectively. Among the three methods Gini index provides slightly better accuracy than the other two methods. All of them can achieve zero error when ϵ is smaller than 0.001. The results shown here align with the count accuracy shown in Table 3.4.

Finally, we check the ranking accuracy of each of the three online methods with respect to their offline version. In Table 3.6 is shown that the ranking error can be eliminated when using ϵ smaller than 0.01, even for the most difficult datasets. The features compared here are top 10% ranked features according to scores calculated by the offline methods. 10 percent of all features is usually the zone where most important features reside. Therefore, for the sole purpose of screening feature selection, using online methods with an relative large ϵ is enough to assure that we can get identical feature ranking results as the offline methods.

Table 3.5: Influence of ϵ on accuracy (3.19) of online chi-square/gini index/mutual information scores compared to their offline versions

epsilon	url	SMK_CAN_187	dexter	dorothea	gisette	kdd12	madelon
Chi-square							
0.2	6.60E-06	0.065	4.20E-05	0	0.001	0	0.18
0.02	2.90E-07	0	6.60E-07	0	4.30E-05	0	0.0042
0.01	4.90E-08	0	4.70E-07	0	1.00E-05	0	0.00089
0.002	8.30E-09	0	0	0	0	0	0
0.001	2.00E-09	0	0	0	0	0	0
0.00066667	0	0	0	0	0	0	0
0.0005	0	0	0	0	0	0	0
Gini index							
0.2	5.40E-06	0.064	3.40E-05	0	0.00037	0	0.12
0.02	1.20E-07	0	1.40E-06	0	2.40E-05	0	0.0029
0.01	4.30E-08	0	2.20E-07	0	3.20E-06	0	0.00055
0.002	9.00E-09	0	0	0	0	0	0
0.001	0	0	0	0	0	0	0
0.00066667	0	0	0	0	0	0	0
0.0005	0	0	0	0	0	0	0
Mutual information							
0.2	6.40E-06	0.061	4.50E-05	0	0.00099	0	0.17
0.02	2.70E-07	0	7.70E-07	0	4.30E-05	0	0.0042
0.01	4.50E-08	0	4.00E-07	0	1.10E-05	0	0.00088
0.002	8.40E-09	0	0	0	0	0	0
0.001	2.00E-09	0	0	0	0	0	0
0.00066667	0	0	0	0	0	0	0
0.0005	0	0	0	0	0	0	0

3.4.2 Online Screening Methods with Model Adaptation

In his section, online screening methods with model adaptation capability are evaluated for their performance in handling data with the concept drifting property. Synthetic datasets are generated to provide a measurable reference.

Synthetic data generation. The ground truth is assume to have linear relationship with the features. For the i-th observation, let y_i denote target value and $\mathbf{x}_i$ denote the feature vector associated with each target.

$$y_i = \boldsymbol{\beta}_i \cdot \mathbf{x}_i^T + c + e_i \tag{3.20}$$

Table 3.6: Influence of ϵ on unmatched ranking among top 10% features for chi-square/gini index/mutual information score according to its offline feature ranking

epsilon	url	SMK_CAN_187	dexter	dorothea	gisette	kdd12	madelon
Chi-square							
0.2	0.017	1	0.89	0	0.81	0	1
0.02	0.0019	0	0	0	0.36	0	0.8
0.01	0	0	0	0	0.12	0	0.42
0.002	0	0	0	0	0	0	0
0.001	0	0	0	0	0	0	0
0.00066667	0	0	0	0	0	0	0
0.0005	0	0	0	0	0	0	0
Gini index							
0.2	0.044	1	0.88	0	0.66	0	0.92
0.02	0.00054	0	0	0	0.28	0	0.68
0.01	0.00027	0	0	0	0.028	0	0.28
0.002	0	0	0	0	0	0	0
0.001	0	0	0	0	0	0	0
0.00066667	0	0	0	0	0	0	0
0.0005	0	0	0	0	0	0	0
Mutual information							
0.2	0.018	1	0.92	0	0.8	0	1
0.02	0.0026	0	0	0	0.39	0	0.8
0.01	0.0004	0	0	0	0.12	0	0.44
0.002	0	0	0	0	0	0	0
0.001	0	0	0	0	0	0	0
0.00066667	0	0	0	0	0	0	0
0.0005	0	0	0	0	0	0	0

Equation 3.20 illustrates the relation between the target and the features. Here $\boldsymbol{\beta}_i$ is a coefficient vector with each of its elements corresponding to the elements in $\mathbf{x}_i$, c is a constant, e_i is a random noise $e_i \sim N(0,1)$. The feature vector $\mathbf{x}_i$ is generated as follow:

$$\mathbf{x}_i = \nu z_i \mathbf{o} + \widetilde{\boldsymbol{\epsilon}}_i \tag{3.21}$$

Where $z_i \sim N(0,1)$, $\mathbf{o}$ is a vector of the same length as $\mathbf{x}_i$ and all its entries equal to 1, ν is a parameter to control the correlation between features, and $\widetilde{\boldsymbol{\epsilon}}_i$ is a noise vector with its j-th element $\widetilde{e_{ij}} \sim N(0,1)$. In this setup, the correlation between any two features is $\nu^2/(1+\nu^2)$. In our experiment ν is set to $\nu = 0.5$, so any two variables have correlation 0.2.

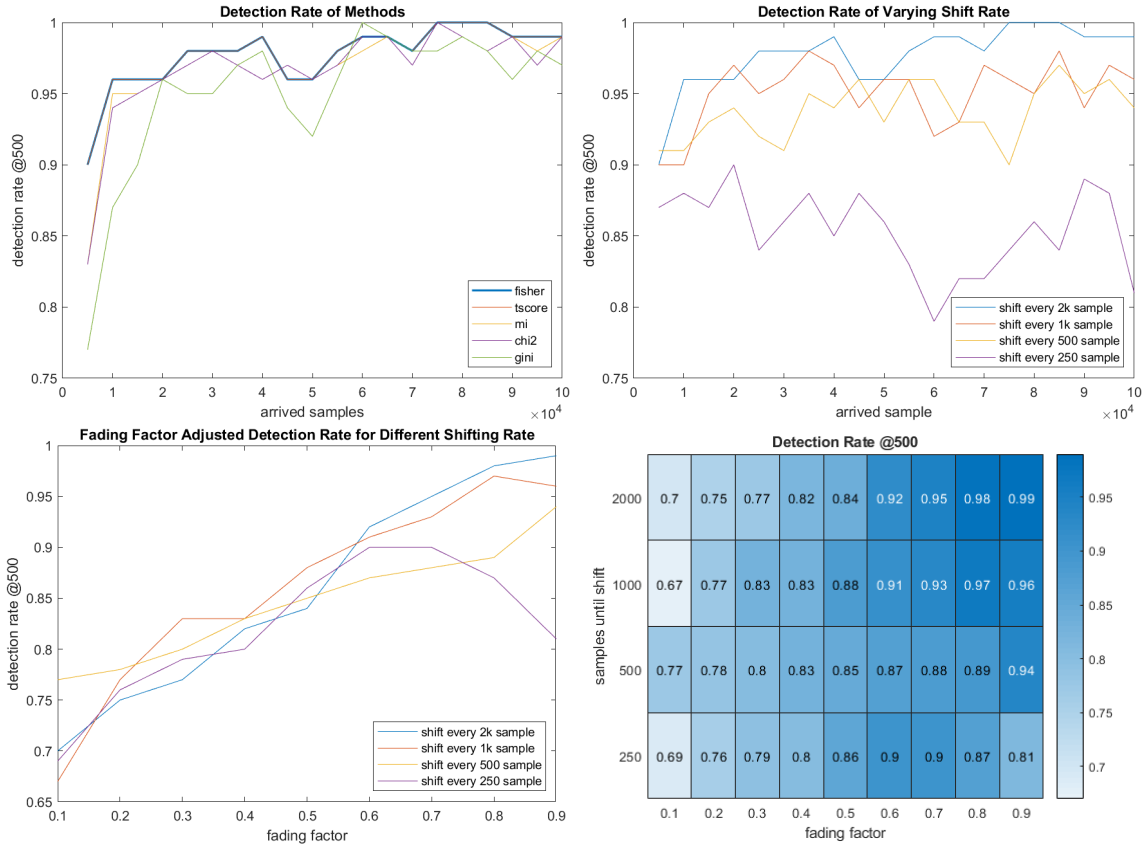

Figure 3.2: Top-left: detection rate by methods. Top-right: detection rate vs shifting rate. Bottom-left: detection rate by shifting rate adjusted by fading factor. Bottom-right: detection rate by shifting rate vs. fading factor.

Let j denote the element index in $\boldsymbol{\beta}_i$ and k denote the number of non-zero coefficients. The indices of non-zero coefficients are shifted every l observations. Let b denote the non-zero signal value. Then $\boldsymbol{\beta}_i$ is constructed as:

$$\beta_{ij} = \begin{cases} b & \text{if } j \in (\lfloor \frac{i}{l} \rfloor : \lfloor \frac{i}{l} \rfloor + k) \\ 0 & \text{otherwise} \end{cases} \tag{3.22}$$

In our experiments, we choose feature space to be 1000D, thus $\mathbf{x}_i \in \mathbb{R}^{1000}$. The number of true features k is set to 100. The coefficient signal value is set to $b = 1$. A total of 100,000 samples are generated. We change the value of l (number of samples until shift) to control the concept drifting level.

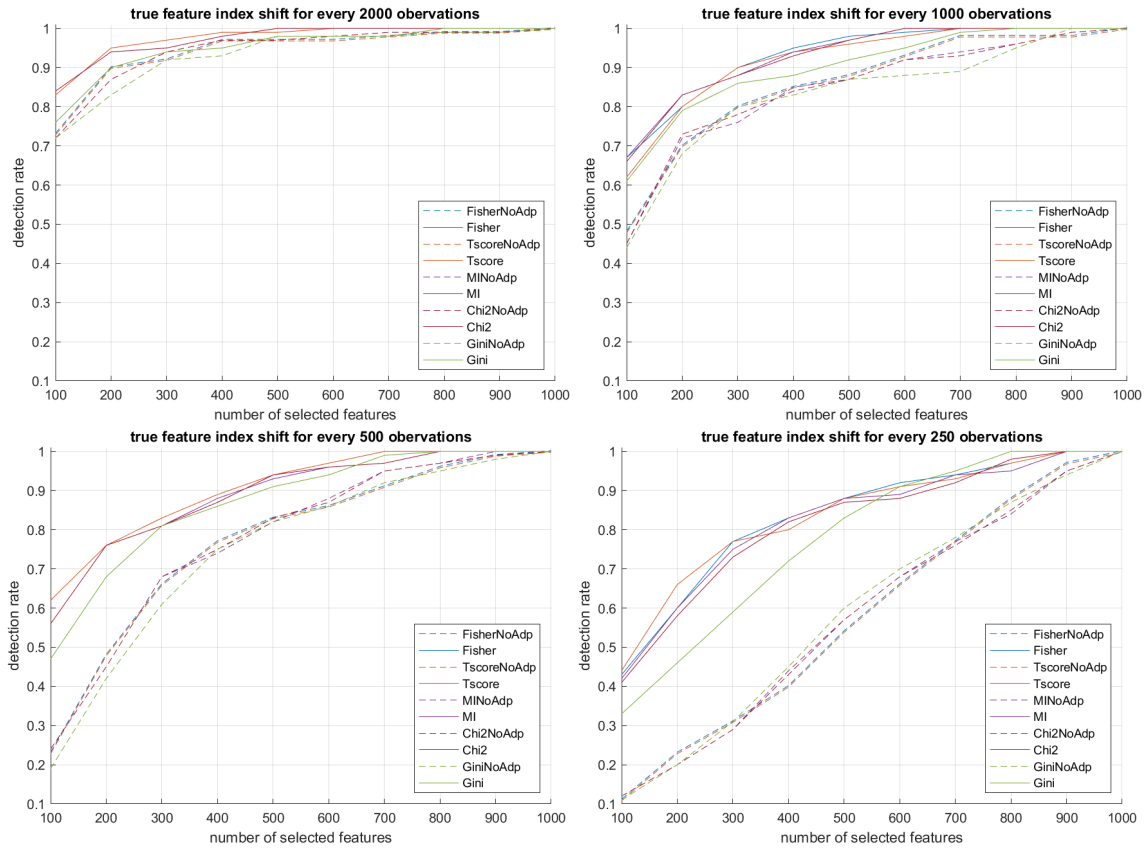

Figure 3.3: Influence of model adaptation on true variable detection rates for different rates of concept drift. Solid curves denote methods with model adaptation, dashed curves are methods without model adaptation.

Results. In this section, we use detection rate @k to measure how well the algorithm performed. Given a set of k selected feature indices FS and a set of true feature indices TU, the detection rate @k is defined as.

$$DetRate@k = \frac{|FS \cap TU|}{|TU|} \tag{3.23}$$

All k-s in this section are set to 500.

Performance Plots. Fig 3.2 visualizes the performance of online screening methods handling data with concept drifting property. Top-left graph shows the overall performance of different screening methods. The shifting rate l is fixed to 2000, and the fading factor α is fixed to $\alpha = 0.9$. The five screening methods show very similar performance. However compared to the others, the Gini Index requires more samples to establish a good performance at the beginning. The graph on the top-right shows how shifting rate affects the variable detection rate. The fading factor α is still

fixed to $\alpha = 0.9$. It is obvious that a fast shifting rate has a negative impact on the performance of the algorithm. On the bottom-left graph, it can be observed that by adjusting the fading factor α, the detection rate can be improved even for data with a faster shifting rate. The heatmap on the bottom-right shows the impact of fading factor on detection rate in detail.

In Fig 3.3, it is shown in detail how the detection rate changes when different number of features are selected by online screening methods, as well as whether model adaptation helps improve the performance of the screening methods. Across all four shift rates that are tested, the screening methods with model adaptation outperform those without model adaptation. Generally, the slower the shift the fewer features need to be selected to provide a full detection. Even in the fastest shift rate that is tested (shift every 250 samples), the screening methods with model adaptation still manage to detect all true features with fewer selected features and consistently detect more true features than the methods without model adaptation.

3.4.3 Realistic Performance of Online Screening Methods

This section provides an insight of the performance of online screening methods on real world large datasets. Different learners and screening methods are combined to generate prediction. The misclassification error rate is used in this section to measure the performance of the algorithm. For learners, Sparse FSA and SGD (stochastic gradient decent with log loss) are chosen.

Table 3.7: The datasets for realistic evaluation

Dataset	Learning type	Feature type	Number of features	Number of observations
20NewsGroups [Keerthi et al., 2005]	Classification	Binary	723,066	11,862
Url [Ma et al., 2009]	Classification	Continuous, Binary	3.2million	2million

Data sets. Among the datasets showed in Table 3.7, Url data already be introduced in Section 3.4.1. Here Url data from day 0 to 99 is used to train the model. The 20NewsGroups is an email content data that is generated according to [Keerthi et al., 2005]. The data originates from the UCI repository [Dua and Graff, 2017]. We also extracted the timestamp for each email and sorted the data in time order.

Results. For all tables in this section, the cell value "random select" indicates that a specified number of features are randomly selected as input for the learner. The first two rows in each table

show the computing time for online screening methods. The "Run Time" column contains the training time only.

In Table 3.8 is shown that for the 20NewsGroup data, some screening methods such as Mutual Information provide improvement to Sparse FSA. Moreover, in most cases, the screening methods with model adaptation give better performance than without model adaptation. For SGD, there is slight improvement given by bin-count based screening methods.

In Table 3.9, it is shown that there is no improvement by applying screening methods on Sparse FSA. The training time is reduced due to smaller feature space caused by screening methods. When combined with SGD, the mean-variance based screening methods with model adaptation provide significant improvement on performance.

Table 3.8: 20 News Groups

Adaptation	Screening Method	Selected Features	Learner	Learner Selected	Run Time Seconds	Error Rate
1	Quantile based				7.21	
1	Moving average				3.75	
0			sparseFSA	35k	37.86	0.064
0	random select	70k	sparseFSA	35k	13.42	0.144
1	T-score	70k	sparseFSA	35k	39.38	0.062
0	T-score	70k	sparseFSA	35k	40.3	0.062
1	Fisher	70k	sparseFSA	35k	48.31	0.063
0	Fisher	70k	sparseFSA	35k	47.87	0.062
1	MI	70k	sparseFSA	35k	39.42	0.061
0	MI	70k	sparseFSA	35k	39.51	0.062
1	Chi2	70k	sparseFSA	35k	40.27	0.063
0	Chi2	70k	sparseFSA	35k	41.05	0.062
1	Gini	70k	sparseFSA	35k	40.63	0.062
0	Gini	70k	sparseFSA	35k	40.87	0.062
0			SGD		0.299	0.063
0	random select	35k	SGD		0.324	0.172
1	T-score	35k	SGD		0.333	0.064
0	T-score	35k	SGD		0.349	0.064
1	Fisher	35k	SGD		0.174	0.066
0	Fisher	35k	SGD		0.146	0.065
1	MI	35k	SGD		0.306	0.062
0	MI	35k	SGD		0.344	0.064
1	Chi2	35k	SGD		0.33	0.062
0	Chi2	35k	SGD		0.385	0.062
1	Gini	35k	SGD		0.335	0.063
0	Gini	35k	SGD		0.351	0.062

Table 3.9: URL data

Adaptation	Screening Method	Selected Features	Learner	Learner Selected	Run Time Seconds	Error Rate
1	Quantile based				205	
1	Moving average				142	
0			SparseFSA	50K	12484	0.0091
0	Random Index	1000K	SparseFSA	50K		0.0155
1	MI	1000K	SparseFSA	50K	7055	0.0097
0	MI	1000K	SparseFSA	50K	7173	0.0104
1	Chi2	1000K	SparseFSA	50K	7674	0.0106
0	Chi2	1000K	SparseFSA	50K	7813	0.0122
1	Gini	1000K	SparseFSA	50K	6612	0.0101
0	Gini	1000K	SparseFSA	50K	7250	0.0122
1	Fisher	1000K	SparseFSA	50K	7938	0.012
0	Fisher	1000K	SparseFSA	50K	7842	0.0122
1	T-score	1000K	SparseFSA	50K	7887	0.0102
0	T-score	1000K	SparseFSA	50K	7819	0.013
0			SGD		147	0.008
0	Random Index	2000K	SGD			0.0085
1	T-score	2000K	SGD		102	0.0073
0	T-score	2000K	SGD		112	0.0097
1	Fisher	2000K	SGD		102	0.0072
0	Fisher	2000K	SGD		118	0.0092
1	MI	2000K	SGD		63	0.0093
0	MI	2000K	SGD		64	0.0095
1	Chi2	2000K	SGD		63	0.0093
0	Chi2	2000K	SGD		65	0.0092
1	Gini	2000K	SGD		64	0.0093
0	Gini	2000K	SGD		64	0.0092

CHAPTER 4

CONCLUSION AND FUTURE WORK

4.1 Conclusion

Since we are interested in evaluating screening methods on real datasets, we don't have information about the true features that are relevant in connection with the response, so we can only look at prediction performance. In this respect, there are at least two ways to see whether the screening methods are useful for real datasets.

If we ask whether they are helpful in improving the prediction performance of the best learning algorithm from our arsenal, then the answer is "Some of them are sometimes useful, on two datasets out of five, in both regression and classification". Indeed, for regression we see from Table 2.3 that Mutual Information and RReliefF were helpful in improving the prediction of the best learning algorithm on two datasets out of five, while the other two screening methods were only helpful on one dataset. For classification, we see from Table 2.7 that most screening methods were helpful in improving the prediction of the best learning algorithm on two datasets out of five, except Relief, which was only helpful on one dataset.

If however we are interested in using a screening method to reduce the dataset size, then we might ask whether we lose any prediction performance this way. In this case our answer would be "Usually not, for the right screening method, especially in classification". In Table 2.3 and Table 2.7 are shown that only in very few occasions do the screening methods harm the performance of best learning algorithm. For regression, we see from Table 2.5 that RReliefF is the best in this respect, remaining in the top performing group (with the right algorithm and number of selected features) on 4 out of 5 regression datasets. For classification, from Table 2.9 we see that Mutual Information and Gini index are the best, remaining in the top performing group (with the right algorithm and number of selected features) on 4 out of 5 classification datasets.

If we had to select one screening method that is most successful at both of these tasks, this method would be Mutual Information. We see that it is the only method that is helpful in improving

performance in both regression and classification, and stays in the top performing group on most datasets, for both regression and classification.

Some of the screening methods that were evaluated in Chapter 2 bring an improvement in prediction for some datasets, in both regression and classification. In the classification tasks, the screening methods with boosted trees give the best overall results. All the seven classification screening methods evaluated helped improve the performance of the learner to a certain degree. The Mutual Information, Gini Index, Chi-square score and Relief work slightly better than the other methods. It also can be seen from the tables that the screening methods work well especially on learning algorithms that give poor results on their own. Compared to classification, there are fewer screening methods for regression problems. Of the three regression screening methods evaluated, RReliefF and Mutual Information work better than correlation, and improve the best learning algorithm performance on two datasets out of five.

In our studies of online screening methods, the moving average based online screening methods can be proved to have the same performance as their offline version and have the advantage of a faster speed and lower storage requirement. The experiments in Section 3.4.1 show that the bin-count based online screening methods can also achieve the same results as their offline version given the right ϵ. Moreover, they can obtain faster or about the same computation speed compared to their offline versions with the right combination of ϵ and minibatch size.

The results in Section 3.4.2 give empirical evidences that adding model adaptation utility to screening methods can help improve their performance when data have concept drifting property. It is shown that to some degree, adjusting fading factor can assist screening method to tackle datasets with high concept drifting rate. The real data analysis in Section 3.4.3 further demonstrates the capability of online screening methods in dealing with real life large datasets with sparsity and possible concept drifting.

Our results show that online screening methods with model adaptation is computationally efficient. They are useful in improving the model performance of complex data learning, especially when sample size and feature space are extremely large.

4.2 Future Work

Some criteria used by screening methods such as mutual information and Gini index are also applied in learning algorithms such as decision tree as a component to judge impurity. It triggers our interests to study whether online quantile based methods can improve the speed or even performance of decision trees.

Throughout the years some works were done to describe the reconstruction and transformation from decision trees to neural network. Several mapping methods were mentioned in [Sethi, 1990] [Sethi, 1991] [Kontschieder et al., 2015] and [Ioannou et al., 2016]. A new framework Neural Rule Ensembles (NRE) introduced in recent literature [Dawer et al., 2020] also focused on such a mapping strategy. It shows that any decision tree can be mapped into a set of neural rules, and the ensemble of neural rules can subsequently be trained using back-propagation. Therefore the expected improvement of training speed on decision trees can hopefully also reflect on neural network training.

BIBLIOGRAPHY

John C Davis and Robert J Sampson. *Statistics and data analysis in geology*, volume 646. Wiley New York et al., 1986.

David D Lewis. Feature selection and feature extraction for text categorization. In *Proceedings of the workshop on Speech and Natural Language*, pages 212–217. Association for Computational Linguistics, 1992.

Kenji Kira and Larry A Rendell. The feature selection problem: Traditional methods and a new algorithm. In *AAAI*, volume 2, pages 129–134, 1992.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

CP Han, D Chris, and HL Fu. Minimum redundancy maximum relevance feature selection [j]. *IEEE Intelligent Systems*, 20(6):70–71, 2005.

Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P Trevino, Jiliang Tang, and Huan Liu. Feature selection: A data perspective. *ACM Computing Surveys (CSUR)*, 50(6): 94, 2017a.

Jiliang Tang, Salem Alelyani, and Huan Liu. Feature selection for classification: A review. *Data Classification: Algorithms and Applications*, page 37, 2014.

Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

Alan Jović, Karla Brkić, and Nikola Bogunović. A review of feature selection methods with applications. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1200–1205. IEEE, 2015.

Jie Cai, Jiawei Luo, Shulin Wang, and Sheng Yang. Feature selection in machine learning: A new perspective. *Neurocomputing*, 300:70–79, 2018.

Yun Li, Tao Li, and Huan Liu. Recent advances in feature selection and its applications. *Knowledge and Information Systems*, 53(3):551–577, 2017b.

Ryan J Urbanowicz, Randal S Olson, Peter Schmitt, Melissa Meeker, and Jason H Moore. Benchmarking relief-based feature selection methods for bioinformatics data mining. *Journal of biomedical informatics*, 85:168–188, 2018.

Salem Alelyani, Jiliang Tang, and Huan Liu. Feature selection for clustering: A review. *Data Clustering: Algorithms and Applications*, 29:110–121, 2013.

Luis Talavera. An evaluation of filter and wrapper methods for feature selection in categorical clustering. *Advances in Intelligent Data Analysis VI*, pages 742–742, 2005.

Yosef Masoudi-Sobhanzadeh, Habib Motieghader, and Ali Masoudi-Nejad. Featureselect: a software for feature selection based on machine learning approaches. *BMC bioinformatics*, 20(1):170, 2019.

Zhen Chen, Pei Zhao, Fuyi Li, André Leier, Tatiana T Marquez-Lago, Yanan Wang, Geoffrey I Webb, A Ian Smith, Roger J Daly, Kuo-Chen Chou, et al. ifeature: a python package and web server for features extraction and selection from protein and peptide sequences. *Bioinformatics*, 34(14):2499–2502, 2018.

Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

Noelia Sánchez-Maroño, Amparo Alonso-Betanzos, and María Tombilla-Sanromán. Filter methods for feature selection–a comparative study. *Intelligent Data Engineering and Automated Learning-IDEAL 2007*, pages 178–187, 2007.

Yvan Saeys, Iñaki Inza, and Pedro Larrañaga. A review of feature selection techniques in bioinformatics. *bioinformatics*, 23(19):2507–2517, 2007.

Marko Robnik-Šikonja and Igor Kononenko. An adaptation of relief for attribute estimation in regression. In *Machine Learning: Proceedings of the Fourteenth International Conference (ICML'97)*, pages 296–304, 1997.

Adrian Barbu, Yiyuan She, Liangjing Ding, and Gary Gramajo. Feature selection with annealing for computer vision and big data learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(2):272–286, 2017.

Huan Liu and Rudy Setiono. Chi2: Feature selection and discretization of numeric attributes. In *Tools with artificial intelligence, 1995. proceedings., seventh international conference on*, pages 388–391. IEEE, 1995.

Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

CW Gini. Variability and mutability, contribution to the study of statistical distribution and relaitons. *Studi Economico-Giuricici della R*, 1912.

Susanna Wang, Nadir Yehya, Eric E Schadt, Hui Wang, Thomas A Drake, and Aldons J Lusis. Genetic and genomic analysis of a fat mass trait with complex inheritance reveals marked sex specificity. *PLoS genetics*, 2(2):e15, 2006.

Robert L. Grossman, Allison P. Heath, Vincent Ferretti, Harold E. Varmus, Warren A. Lowy, Douglas R.and Kibbe, and Louis M. Staudt. Toward a shared vision for cancer genomic data. *New England Journal of Medicine*, 375(12):1109–1112, 2016.

Joaquín Torres-Sospedra, Raúl Montoliu, Adolfo Martínez-Usó, Joan P Avariento, Tomás J Arnau, Mauri Benedito-Bordonau, and Joaquín Huerta. Ujiindoorloc: A new multi-building and multi-floor database for wlan fingerprint-based indoor localization problems. In *Indoor Positioning and Indoor Navigation (IPIN), 2014 International Conference on*, pages 261–270. IEEE, 2014.

Rasmus Rothe, Radu Timofte, and Luc Van Gool. Deep expectation of real and apparent age from a single image without facial landmarks. *International Journal of Computer Vision (IJCV)*, July 2016.

Ovidiu Ivanciuc. CoEPrA 2006 Round 3 comparative evaluation of prediction algorithms, 2006. URL http://www.coepra.org/.

Isabelle Guyon, Steve Gunn, Asa Ben-Hur, and Gideon Dror. Result analysis of the nips 2003 feature selection challenge. In *Advances in neural information processing systems*, pages 545–552, 2005.

Avrum Spira, Jennifer E Beane, Vishal Shah, Katrina Steiling, Gang Liu, Frank Schembri, Sean Gilman, Yves-Martine Dumas, Paul Calner, Paola Sebastiani, et al. Airway epithelial gene expression in the diagnostic evaluation of smokers with suspect lung cancer. *Nature medicine*, 13(3):361, 2007.

William A Freije, F Edmundo Castro-Vargas, Zixing Fang, Steve Horvath, Timothy Cloughesy, Linda M Liau, Paul S Mischel, and Stanley F Nelson. Gene expression profiling of gliomas strongly predicts survival. *Cancer research*, 64(18):6503–6510, 2004.

M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, et al. Deep face recognition. In *BMVC*, volume 1, page 6, 2015.

Matlab release 2016b, 2016. The MathWorks, Inc., Natick, MA, USA.

Xuan Vinh Nguyen(2014). Information theoretic feature selection, version 1.1, Updated 07 Jul 2014. URL https://www.mathworks.com/matlabcentral/fileexchange/47129-information-theoretic-feature-selection.

Mingyuan Wang and Adrian Barbu. Are screening methods useful in feature selection? an empirical study. *PloS one*, 14(9):e0220842, 2019.

José Ramon Méndez, Florentino Fdez-Riverola, Daniel Glez-Peña, Fernando Díaz, and Juan M Corchado. Relaxing feature selection in spam filtering by using case-based reasoning systems. In *Portuguese Conference on Artificial Intelligence*, pages 53–62. Springer, 2007.

Vitor R Carvalho and William W Cohen. Single-pass online learning: Performance, voting schemes and online feature selection. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 548–553, 2006.

Jialei Wang, Peilin Zhao, Steven CH Hoi, and Rong Jin. Online feature selection and its applications. *IEEE Transactions on Knowledge and Data Engineering*, 26(3):698–710, 2013.

Ioannis Katakis, Grigorios Tsoumakas, and Ioannis P Vlahavas. On the utility of incremental feature selection for the classification of textual data streams. In *Panhellenic Conference on Informatics*, volume 5, pages 338–348. Citeseer, 2005.

Mohammad M Masud, Qing Chen, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani Thuraisingham. Classification and novel class detection of data streams in a dynamic feature space. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 337–352. Springer, 2010.

Joao Bartolo Gomes, Mohamed Medhat Gaber, Pedro AC Sousa, and Ernestina Menasalvas. Mining recurring concepts in a dynamic feature space. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1):95–110, 2013.

Blaz Sovdat. Updating formulas and algorithms for computing entropy and gini index from time-changing data streams. *arXiv preprint arXiv:1403.6348*, 2014.

Márcia Henke, Eduardo Souto, and Eulanda M dos Santos. Analysis of the evolution of features in classification problems with concept drift: Application to spam detection. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 874–877. IEEE, 2015.

Fabian Keller, Emmanuel Müller, and Klemens Böhm. Estimating mutual information on data streams. In *Proceedings of the 27th International Conference on Scientific and Statistical Database Management*, pages 1–12, 2015.

Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical review E*, 69(6):066138, 2004.

Mahmood Shakir Hammoodi, Frederic Stahl, and Atta Badii. Real-time feature selection technique with concept drift detection using adaptive micro-clusters for data stream mining. *Knowledge-Based Systems*, 161:205–239, 2018.

J Ian Munro and Mike S Paterson. Selection and sorting with limited storage. *Theoretical computer science*, 12(3):315–323, 1980.

Raj Jain and Imrich Chlamtac. The p 2 algorithm for dynamic calculation of quantiles and histograms without storing observations. *Communications of the ACM*, 28(10):1076–1085, 1985.

Rakesh Agrawal and Arun N. Swami. A one-pass space-efficient algorithm for finding quantiles. In *COMAD*, 1995.

Joao Gama and Carlos Pinto. Discretization from data streams: applications to histograms and data mining. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 662–667, 2006.

Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G Lindsay. Approximate medians and other quantiles in one pass and with limited memory. *ACM SIGMOD Record*, 27(2):426–435, 1998.

Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. In *ACM SIGMOD Record*, volume 28, pages 251–262. ACM, 1999.

Michael Greenwald, Sanjeev Khanna, et al. Space-efficient online computation of quantile summaries. *ACM SIGMOD Record*, 30(2):58–66, 2001.

Qi Zhang and Wei Wang. A fast algorithm for approximate quantiles in high speed data streams. In *19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)*, pages 29–29. IEEE, 2007.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

Xuemin Lin, Hongjun Lu, Jian Xu, and Jeffrey Xu Yu. Continuously maintaining quantile summaries of the most recent n elements over a data stream. In *Proceedings. 20th International Conference on Data Engineering*, pages 362–373. IEEE, 2004.

Graham Cormode, Flip Korn, Shan Muthukrishnan, and Divesh Srivastava. Space-and time-efficient deterministic algorithms for biased quantiles over data streams. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 263–272. ACM, 2006.

Michael B Greenwald and Sanjeev Khanna. Power-conserving computation of order-statistics over sensor networks. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 275–285. ACM, 2004.

Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 239–249. ACM, 2004.

Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Identifying suspicious urls: an application of large-scale online learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 681–688. ACM, 2009.

Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. Field-aware factorization machines for ctr prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 43–50. ACM, 2016.

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

Matlab release 2018b, 2018. The MathWorks, Inc., Natick, MA, USA.

S Sathiya Keerthi, Dennis DeCoste, and Thorsten Joachims. A modified finite newton method for fast solution of large scale linear svms. *Journal of Machine Learning Research*, 6(3), 2005.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Ishwar Krishnan Sethi. Entropy nets: from decision trees to neural networks. *Proceedings of the IEEE*, 78(10):1605–1613, 1990.

Ishwar K Sethi. Decision tree performance enhancement using an artificial neural network implementation. In *Machine Intelligence and Pattern Recognition*, volume 11, pages 71–88. Elsevier, 1991.

Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep neural decision forests. In *Proceedings of the IEEE international conference on computer vision*, pages 1467–1475, 2015.

Yani Ioannou, Duncan Robertson, Darko Zikic, Peter Kontschieder, Jamie Shotton, Matthew Brown, and Antonio Criminisi. Decision forests, convolutional networks and the models in-between. *arXiv preprint arXiv:1603.01250*, 2016.

Gitesh Dawer, Yangzi Guo, Sida Liu, and Adrian Barbu. Neural rule ensembles: Encoding sparse feature interactions into neural networks. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020.

BIOGRAPHICAL SKETCH

Mingyuan Wang received his Bachelor's degree in Software Engineering in 2013. In 2014, after he obtained his M.A. degree in Financial Economics from University of Detroit Mercy. He started his education in statistics at Florida State University and obtained his M.S. degree in Statistics in 2017.

He started his PhD program in Statistics at Florida State University the same year under the supervision of Dr. Adrian Barbu. He worked on researches related to conventional screening feature selection method and online feature screening method which tackles the stream processing of huge sparse data. He also worked on projects in medical image/video processing and electrical signal processing.