Data Split Strategies for Evolving Predictive Models

Vikas C. Raykar(✉) and Amrita Saha

IBM Research, Bangalore, India
{viraykar,amrsaha4}@in.ibm.com

Abstract. A conventional textbook prescription for building good predictive models is to split the data into three parts: training set (for model fitting), validation set (for model selection), and test set (for final model assessment). Predictive models can potentially evolve over time as developers improve their performance either by acquiring new data or improving the existing model. The main contribution of this paper is to discuss problems encountered and propose workflows to manage the allocation of newly acquired data into different sets in such dynamic model building and updating scenarios. Specifically we propose three different workflows (parallel dump, serial waterfall, and hybrid) for allocating new data into the existing training, validation, and test splits. Particular emphasis is laid on avoiding the bias due to the repeated use of the existing validation or the test set.

Keywords: Data splits · Model assessment · Predictive models

1 Introduction

A common data mining task is to build a good predictive model which generalizes well on future unseen data. Based on the annotated data collected so far the goal for a machine learning practitioner is to search for the best predictive model (known as *supervised learning*) and at the same time have a reasonably good estimate of the performance (or risk) of the model on future unseen data. It is well known that the performance of the model on the data used to learn the model (training set) is an overly optimistic estimate of the performance on unseen data. For this reason it is a common practice to sequester a portion of the data to assess the model performance and never use it during the actual model building process. When we are in a data rich situation a conventional textbook prescription (for example refer to Chapter 7 in [6]) is to split the data into three parts: *training set*, *validation set*, and *test set* (See Figure 1). The training set is used for *model fitting*, that is, estimate the parameters of the model. The validation set is used for *model selection*, that is, we use the performance of the model on the validation set to select among various competing models (e.g. should we use a linear classifier like logistic regression or a non-linear neural network) or to choose the hyperparameters of the model (e.g. choosing the regularization

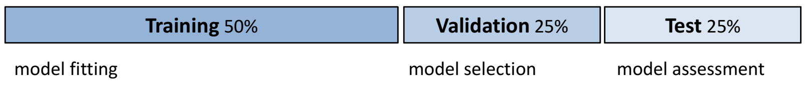


Fig. 1. *Data splits for model fitting, selection, and assessment.* The *training split* is used to estimate the model parameters. The *validation split* is used to estimate prediction error for model selection. The *test split* is used to estimate the performance of the final chosen model.

parameter for logistic regression or the number of nodes in the hidden layer for a neural network). The test set is then used for final *model assessment*, that is, to estimate the performance of the estimated model.

However in practice searching for the best predictive model is often an iterative and continuous process. A major bottleneck typically encountered in many learning tasks is to collect the data and annotate them. Due to various constraints (either time or financial) very often the best model based on the data available so far is deployed in practice. At the same time the data collection and annotation process will continue so that the model can be improved at a later stage. Once we have reasonably enough data we refit the model to the new data to make it more accurate and then release this new model. Sometimes after the model has been deployed in practice we find that the model does not perform well on a new kind of data which we do not have in our current training set. So we redirect our efforts into collecting more data on which our model fails. The main contribution of this paper is to discuss problems encountered and propose various workflows to manage the allocation of newly acquired data into different sets in such dynamic model building/updating scenarios.

With the advent of increased computing power it is very easy to come up with a model that performs best on the validation set by searching over an extremely large range of diverse models. This procedure can lead to non-trivial bias (or over-fitting to the validation set) in the estimated model parameters. It is very likely that we found the best model on the validation set by chance. The same applies to the testing set. One way to think of this is that every time we use the test set to estimate the performance the dataset becomes less fresh and can increase the risk of over-fitting. The proposed data allocation workflows are designed with a particular emphasis on avoiding this bias.

2 Data Splits for Model Fitting, Selection, and Assessment

A typical supervised learning scenario consists of an annotated data set $\mathcal{T} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ containing n instances, where $\boldsymbol{x}_i \in \mathcal{X}$ is an instance (typically a d-dimensional feature vector) and $y_i \in \mathcal{Y}$ is the corresponding known label. The task is to learn a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ which performs well on an independent test data and also have a reasonably good estimate of the performance (also known as the test error of the model). Let $\widehat{f}(\boldsymbol{x})$ be the prediction model/function that has been learnt/estimated using the training data $\mathcal{T}$. Let $L(y, \widehat{f}(\boldsymbol{x}))$ be the *loss function* [1] for measuring errors between the target response y and the prediction from the learnt model $\widehat{f}(\boldsymbol{x})$. The (conditional) *test error*, also referred to as *generalization error*, is the prediction error over an independent test sample, that is, $\mathrm{Err}_{\mathcal{T}} = E_{(\boldsymbol{x},y)}[L(y, \widehat{f}(\boldsymbol{x}))|\mathcal{T}]$, where $(\boldsymbol{x}, y)$ are drawn randomly from their joint distribution. Since the training set $\mathcal{T}$ is fixed, and test error refers to the error obtained with this specific training set. Assessment of this test error is very important in practice since it gives us a measure of the quality of the ultimately chosen model (referred to as *model assessment*) and also guides the choice of learning method or model (also known as *model selection*). Typically our model will also have tuning parameters (for example the regularization parameter in lasso or the number of trees in random forest) and we write our predictions as $\widehat{f}_\theta(\boldsymbol{x})$. The tuning parameter θ varies the complexity of our model, and we wish to find the value of θ that minimizes the test error. The *training error* is the average loss over the entire training sample, that is, $\overline{\mathrm{err}} = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \widehat{f}_\theta(\boldsymbol{x}_i))$. Unfortunately *training error is not a good estimate of the test error*. A learning method typically adapts to the training data, and hence the training error will be an overly optimistic estimate of the test error. Training error consistently decreases with model complexity, typically dropping to zero if we increase the model complexity large enough. However, a model with zero training error is overfit to the training data and will typically generalize poorly.

If we are in a data-rich situation, the best approach to estimate the test error is to randomly divide the dataset into three parts [2,4,6]: a *training split* $\mathcal{T}$, a *validation split* $\mathcal{V}$, and a *test split* $\mathcal{U}$. While it is difficult to give a general rule on the split proportions a typical split suggested in [6] is to use 50% for training, and 25% each for validation and testing (see Figure 1). The *training split* $\mathcal{T}$ is used to fit the model (*i.e.* estimate the parameters of the model for a fixed set of tuning parameters). The *validation split* $\mathcal{V}$ is used to estimate prediction error for model selection. We use the performance on the validation split to select among various competing models or to choose the tuning parameters of the model. The *test split* $\mathcal{U}$ is used to estimate the performance of the final chosen model. Ideally, the test set should be sequestered and be brought out only at the end of the data analysis.

In this paper we specifically assume that we are in a *data-rich situation*, that is we have a reasonably large amount of data. In data poor situations where we do not have the luxury of reserving a separate test set, it does not seem possible to estimate conditional error effectively, given only the information in the same training set. A related quantity sometimes used in data poor situations is the expected test error $\mathrm{Err} = E[\mathrm{Err}_{\mathcal{T}}]$. While the estimation of the conditional test error $\mathrm{Err}_{\mathcal{T}}$ will be our goal the expected test Err is more amenable to statistical analysis, and most methods like cross-validation [15] and bootstrap [3] effectively estimate the expected error [6].

[1] Typical loss functions include the 0-1 loss $(L(y, \widehat{f}(\boldsymbol{x})) = I(y \neq \widehat{f}(\boldsymbol{x}))$,where I is the indicator function) or the log-likelihood loss for classification and the squared error $(L(y, \widehat{f}(\boldsymbol{x})) = (y - \widehat{f}(\boldsymbol{x}))^2)$ or the absolute error $(L(y, \widehat{f}(\boldsymbol{x})) = |y - \widehat{f}(\boldsymbol{x})|)$ for regression problems.

3 Issues with Evolving Models

Arrival of New Data After Model Deployment. Having done model selection using the validation split the parameters of the model are estimated using the training split and the performance on unseen data is assessed using the test split. If this performance is reasonable enough the model is finally deployed in practice. This scenario has implicitly assumed that we start the data analysis after all the data is collected. However in practice data collection and annotation is a continuous ongoing process, often in a non-iid fashion. After the final model is frozen and deployed in practice after a few months let us say we have more annotated data. Now the question that arises is how do we use this data. Should we dump all this data into the training split to improve the model performance? Or should we dump this into the test split so that we have a better estimate of the model performance?

Model Driven Data Collection. Very often it so happens that once the model is deployed in practice we discover that the model performs poorly on a certain class of data. The most likely cause of this is that we did not have enough data of that particular kind in our dataset. This drives the collection of specific kind of data on which our model performs poorly. Having collected the data similar questions arise. What options should we pursue for allocating the new data to different splits ?

Test Set Reuse. When developing predictive models in many domains (and especially in medical domains) it is a common practice to completely sequester the test set from the data mining practitioners. Once the model has been finalized a review board (such as the Food and drug administration) evaluates the model and then gives a final decision as to whether the model passed the test or not. The feedback could be binary(pass/fail) or more detailed like the error or the kind of mistakes made. If the model failed the test then they have to go back and build a better model and test it again. With the advent of increased computing power it is very easy to come up with a model that performs best on the test set by searching over an extremely large range of diverse models. In such a scenario multiple reuse of the test set can often lead to overfitting on the test set. The ideal solution is to completely replace the test set, but having a new test set every time can be expensive given that the data mining practitioners keep churning out new models extremely fast. Another approach to avoid this is that the test set has to be kept fresh by supplementing it with new data every time a developer requests it for testing. At the same time the practitioners would like to learn from the mistakes in the test set. This can be achieved by releasing some part of the test set to the practitioners to improve the model.

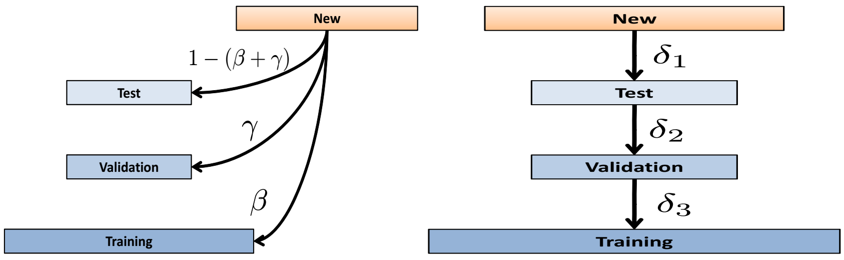
These issues arise across different domains. In natural language processing tasks textual resources are acquired one at a time and annotated. The resource acquisition is often guided by various non-technical constraints and often arrives in a non-iid fashion. In the medical domain a common task is to build a predictive model to predict whether a suspicious region on a medical image is malignant or benign. In order to train such a classifier, a set of medical images is collected from hospitals. These scans are then read by expert radiologists who mark the suspicious locations. Ideally we would like our model to handle all possible variations—different scanners, acquisition protocols, hospitals, and patients. Collecting such data is a time consuming process. Typically we can have contracts with different hospitals to collect and process data. Each hospital has a specific kind of scanners, acquisition protocols, and patient demographics. While most learning methods assume that data is randomly sampled from the population in reality due to various constraints data does not arrive in a random fashion. Based on the contracts at the end of a year we have data from say around five hospitals and the data from the another hospital may arrive a year later. Based on the data from five hospitals we can deploy a model and later update the model when we acquire the data from the other hospital.

These kind of issues also arise in data mining competitions which traditionally operate in a similar setup. Kaggle [1], for example, is a platform for data prediction competitions that allows organizations to post their data and have it scrutinized by thousands of data scientists. The training set along with the labels is released to the public to develop the predictive model. Another set for which the labels have been withheld is used to track the performance of the competitors on a public leader board. Very often is happens that the competitors try to overfit the model on this leader board set. For this reason only a part of this set is used for the leader board and the remaining data is used to decide the final rankings. An important feature of our proposed workflows is that there is a movement of data across different sets at regular intervals and this can help avoid the competitors trying to overfit their models to the leader board.

4 Data Splits for Evolving Models

Based on the data collected so far let us assume we start with a 50% training-25% validation-25% test split as described earlier. In this paper we propose a workflow to allocate newly acquired data into the existing splits. Any workflow to split new data should balance the following desired objectives:

1. ***Exploit large portion for training quickly.*** We want to exploit as much of the new data as quickly as possible for training our final predictive model. For most models, the larger the dataset the more accurate are the estimated parameters.
2. ***Reserve sufficient amount for testing.*** However, at the same time we want to reserve a sufficient amount of the data (in the validation and testing set) for getting an unbiased estimate of the performance of the learnt model. It is very likely that the new data is a different kind of data not existing in the current splits and hence we want to have a sufficient representation of the new data in all the three splits.
3. ***Keep the test set fresh.*** We want to keep the testing/validation sets fresh to avoid the bias due to the reuse of the test set.
4. ***Learn from your mistakes.*** At the same time we do not want to completely sequester the test set and make sure that data mining practitioners



(a) Parallel dump workflow (b) Serial waterfall workflow

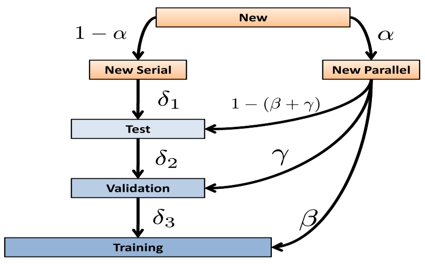
Fig. 2. (a) *Parallel dump workflow* The new data is split into three parts (according to the ratio $\beta : \gamma : 1-(\beta+\gamma)$) and directly dumped into the existing training, validation, and test splits. (b) *Serial waterfall workflow* A δ_3 fraction of the validation set moves to the training set, a δ_2 part of the test set moves to the validation set, and a δ_1 fraction of the new data is allocated to the test set.

learn from our mistakes in the test set. This is especially useful in scenarios where then the data mining practitioners have to go back to their drawing boards and re-design their model because it failed on a sequestered test set.

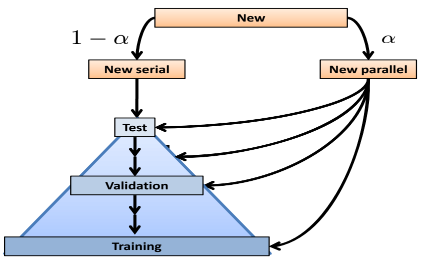
In the next section we describe two workflows: the *parallel dump* (§ 4.1) and the *serial waterfall* (§ 4.2) each of which can address some of these objectives. In § 4.3 we describe the proposed *hybrid workflow* which can balance all the four objectives described above.

4.1 Parallel Dump Workflow

The most obvious method is to split the new data into three parts and directly dump each part into the existing test, validation, and training splits as shown in Figure 2(a). One could use a 50% training-25% validation-25% test split as earlier or any desired ratio $\beta : \gamma : 1-(\beta+\gamma)$. The main advantage of this method is that we have immediate access to the new data in the training set and the model can be improved quickly. Also a sufficient portion of the new data goes into the validation and the test set immediately. However once the new data is allocated we end up using the validation and test splits again one more time (the split is now no longer as fresh as the first time) thus leading to the model selection bias. In this workflow the splits are static and there is no movement across the splits. As a result we do not have a chance to learn from mistakes in the validation and testing set. Generally it may not be to our advantage to let the test split to keep growing without learning from the errors the model makes on the test set. So it makes sense to move some part of the data from the test set to either the training or validation set.



(a) Hybrid workflow



(b) Hybrid continuum workflow

Fig. 3. (a) *The hybrid workflow* There is a movement of data among different sets in the serial mode and at the same time there is an immediate fresh influx of data from the parallel mode. (b) *The continuum workflow* In this setting instead of having 3 sets (training, testing, unseen) we will in theory have a continuum of sets (with samples trickling to the lower levels), with the lowest one for training and the top most one to give the most unbiased estimate of performance.

4.2 Serial Waterfall Workflow

In this workflow data keeps trickling from one level to the other as illustrated in Figure 2(b). Once new data arrives, a δ_3 fraction of the validation set moves to the training set, a δ_2 part of the test set moves to the validation set, and a δ_1 fraction of the new data is allocated to the test set. The training set always keeps getting bigger and once a data moves to the training set it stays there forever. This mode has the following advantages: (1) the test and validation sets are always kept fresh. This avoids over fitting due to extensive model search since the validation and test sets are always refreshed. (2) Since part of the data from the validation and the test set eventually moves to the training we have a chance to learn from the mistakes. The disadvantage of this serial workflow is that the new data takes some time to move to the training set, depending on how often we refresh the existing sets. This restricts us from exploiting the new data as quickly as possible as it takes some time for the data to trickle to the training set.

4.3 Hybrid Workflow

Our proposed workflow as illustrated in Figure 3(a) is a combination of the above two modes of dataflow. It combines the advantages of both these modes. The key feature of the proposed workflow is that there is a movement of data among different sets in the serial mode and at the same time there is an immediate fresh influx of data from the parallel mode. We split the new data randomly into two sets according to the ratio $1 - \alpha : \alpha$. One split is used for the Serial Waterfall mode and the other split is used for the Parallel Dump mode. A value

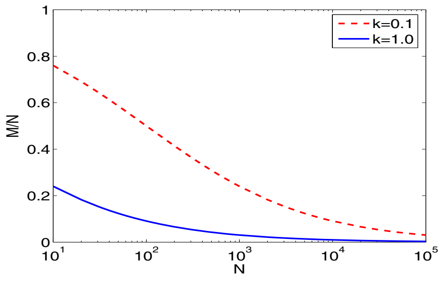


Fig. 4. The ratio $M/N = 1/(k\sqrt{N}+1)$ as a function of N for two different k.

of $\alpha = 0.5$ seems to be a reasonable choice [2]. The value of α can be increased if a more dominant parallel workflow is desired. Note that $\alpha = 0$ corresponds to the serial workflow and $\alpha = 1$ corresponding to the parallel workflow. The value of parameter α is more of design choice and can be chosen based on the domain and various constraints.

In principle the proposed workflow can be extended to more than 3 levels (see Figure 3(b)). In this setting instead of having 3 sets (training, validation, test) we will in theory have a continuum of sets (with samples trickling to the lower levels), with the lowest one to be used for training and the top most one to give the most unbiased estimate of performance of the model. Presumably levels could be: training/tweaking data, cross validation data, hold out testing data, up to highly unbiased very fresh test data. Lower levels would obviously be used more often than higher ones, and testing on higher levels would presumably only happen when tests on lower levels were successful.

5 Bias Due to Test Set Reuse

A key idea in the serial mode is to move data from one level to another to avoid bias due to multiple reuse. We derive a simple rule to decide how much of the new data has to be moved to the test set to avoid the bias due to reuse of the test set multiple times. The same can be used to move data from the test set to the validation set to keep the validation set fresh. This analysis is based on ideas in [14]. Our goal is not to get a exact expression but to get the nature of dependence on the set size. We will consider a scenario where we have a test set of N examples. At each reuse we supplement the test set with M new examples

[2] The parameter α decides the proportion of the incoming data that will go into the train and the test splits. Without any prior knowledge or assumption, the reasonable value of α is a constant fixed at 0.5. But α can be further made to vary every time a new batch of data arrives. For example consider a batch scenario where for every incoming batch of data we can compute the similarity of the current batch with the previously seen batches (for example using the Kullback-Leilbler (KL) divergence). The parameter α can then be made to vary based on the estimated KL divergence.

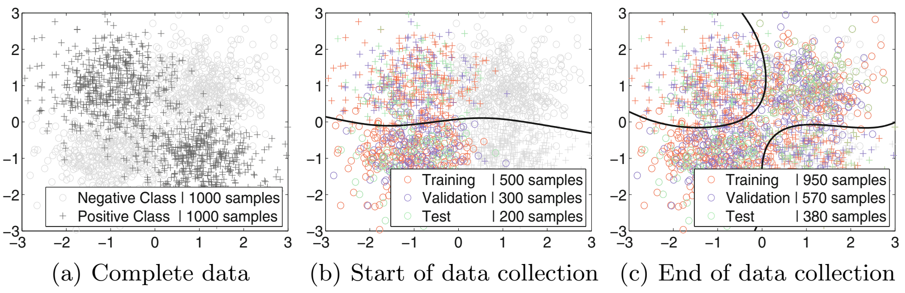


Fig. 5. *Workflow simulation setup* (a) A two-dimensional binary classification simulation setup with data sampled from four Gaussians. (b) The decision boundary obtained by a neural network at the start of the workflow simulation. At each time step we add 50 new examples into the data pool, allocate the new data according to different workflows (either the parallel dump, serial waterfall, or the hybrid workflow) and repeat the model building process. (c) The final decision boundary obtained when all the data has been acquired.

and remove M of the oldest examples. Specifically we will prove the following result (see the appendix for the proof):

After each reuse if we supplement the test set (consisting of N samples) with $M > N/(k\sqrt{N}+1)$ new samples for some small constant k then the bias due to test set reuse can be safely ignored.

Figure 4 plots the ratio M/N as a function of N for two values of k. We need to replace a smaller fraction of the data when N is large. Small datasets lead to a larger bias after each use and hence need to be substantially supplemented.

6 Illustration on Synthetic Data

We first illustrate the advantages and the disadvantages of the three different workflows on a two-dimensional binary classification problem shown in Figure 5(a) with data sampled from a mixture of four Gaussians. The positive class consists of 1000 examples from two Gaussians centered at [-1,1] and [1,-1]. The negative class consists of 1000 examples from a mixture of two Gaussians centered at [1,1] and [-1,-1] respectively. We use a multi-layer neural network with one hidden layer and trained via back propagation. The number of units in the hidden layer is considered as the tuning parameter and selected using the validation split. In order to see the effect of having a non-representative set of data during typical model building scenarios, we consider a scenario where at the beginning of the model building process we have *collected data only from two Gaussians* (positive class centered at [-1,1] and negative class centered at [-1,-1]) as shown in Figure 5(b). Based on the data collected so far we will use the validation split to tune the number of hidden units in the neural network, the training split to train the neural network via back propagation, and the test split to compute the misclassification error of the final trained model. Figure 5(b) shows the decision boundary obtained for the trained neural network using such splits at the start of the model building process. At each time step we add 50 new examples into the data pool, allocate the new data according to different workflows (either the parallel dump, the serial waterfall, or the hybrid workflow) and repeat the model building process. *The new data does not arrive in a random fashion.* We first sample data from the Gaussian centered at [1,1] and then the data from the remaining Gaussian centered at [1,-1]. While this may not be a fully realistic scenario it helps us to illustrate the different workflows. One way to think of this is to visualize that each Gaussian represents data from different hospital/scanner and the data collection may not be designed such that data arrives in a random fashion. Figure 5(c) shows the final decision boundary obtained when all the data has been acquired. Once we have all the data all different workflows reach the same performance. Here we are interested in the model performance and our estimate of it at different stages as new data arrives.

Actual Performance of The Model. Figure 6 plots the misclassification error at each time point (until all the data has been used) for the parallel dump (with parameters $\beta = 0.5$ and $\gamma = 0.3$), the serial waterfall (with δ parameters automatically chosen), and the hybrid workflow (with $\alpha = 0.1$). The error *is computed on the entire dataset* [3]. If we had the entire dataset the final model should have an error of around 0.15. It can be seen that the parallel dump workflow exploits the new data quickly and reaches this performance in around 20 time steps. The serial waterfall moves the data to the training set slowly and achieves the same performance in around 40 time steps. The hybrid workflow can be considered as a compromise between these two workflows and exploits all the new data in around 25 time steps. The sharp drops in the curve occur when the decision boundary changes abruptly because we have now started collecting data from a new cluster.

Estimate of The Performance of The Model. We want to exploit as much of the new data as quickly as possible for training our final predictive model. However at the same time we want to keep the test set fresh in order to get the most unbiased estimate of the performance of the final model. Note that Figure 6 plotted the test error assuming that some oracle gave us the actual data distribution. However in practice we do not have access to this distribution and should use the existing data collected so far (more precisely the test split) to also get a reasonably good estimate of the test error. Figure 7(a), (b), and (c) compares the training error, test error, and the actual error for three different workflows. The results are averaged over 100 repetitions and the standard deviation is also shown. While the test error for all the three workflows approaches the true error when all the data has been collected we want to track how the test error changes during any stage of the model building process. While the

[3] We have shown how to select the delta parameter automatically in the serial waterfall model. The other parameters are more of design choice and have to be chosen based on the various constraints (time, financial, etc.).

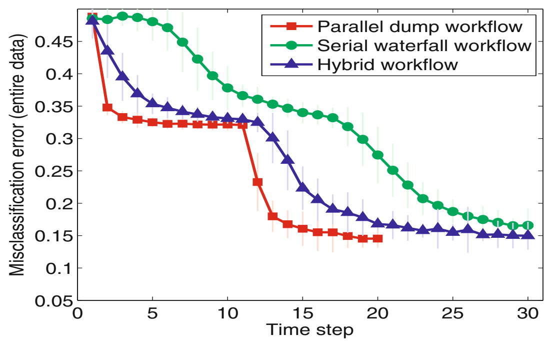
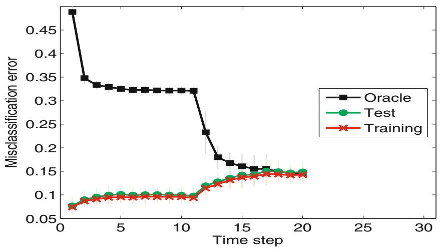


Fig. 6. *Comparison of workflows* The error (computed on the entire dataset) at each time point (until all the data has been used) for the parallel dump (with parameters $\beta = 0.5$ and $\gamma = 0.3$), the serial waterfall (with δ automatically chosen), and the hybrid workflow (with $\alpha = 0.1$).

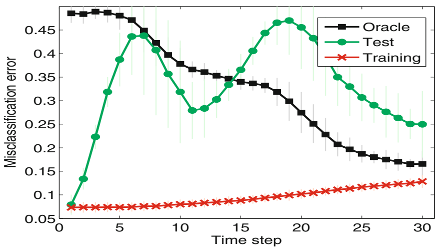
parallel dump workflow (see Figure 7(a)) gave us a predictive model quickly the test error is highly optimistic (and close to the training error) and is close to the true error only at the end when we have access to all the data. The test error for serial waterfall workflow (see Figure 7(b)) does not track the training error and is better reflection of the risk of the model than the parallel dump workflow. These variations in the test error can be explained because the composition of the test set is continuously changed at each time step and it takes some time for the new data to finally reach the training set. However the serial waterfall workflow can be overly pessimistic and the proposed hybrid workflow (see Figure 7(c)) can be a good compromise between the two—it can give a good model reasonably fast and at the same time produce a reasonably good estimate of the performance of the model. By varying the parameter α one can obtain a desired compromise between the serial and the parallel workflow. Figure 7(d) compares the hybrid workflow for two different values of the split parameters $\alpha = 0.1$ and $\alpha = 0.5$ with $\alpha = 0$ corresponding to the serial workflow and $\alpha = 1$ corresponding to the parallel workflow. The value of parameter α is more of design choice and can be chosen based on the domain and various constraints.

7 Case Study: Paraphrase Detection

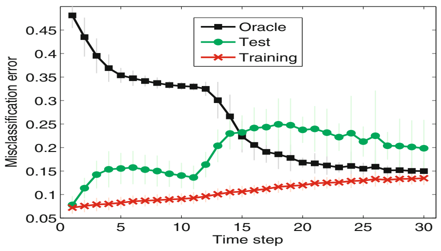
We demonstrate the tradeoffs of the different workflows on a natural language processing task of *paraphrase detection* [8]. Given a pair of sentences, for example, *Video game violence is not related to serious aggressive behavior in real life.* and *Violence in video games is not causally linked with aggressive tendencies.*, we would like to learn a model which predicts whether the two sentences are semantically equivalent (paraphrases) or not. We take a supervised learning approach to this problem by first manually collecting a labeled data, extracting features from the sentences, and then training a binary classifier. Given a pair



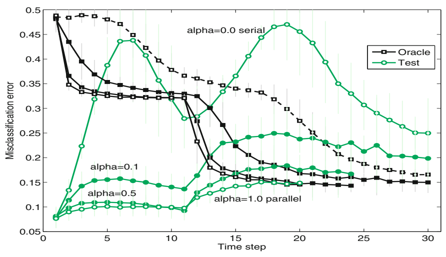
(a) Parallel—Optimistic test error



(b) Serial—Pessimistic test error



(c) Hybrid workflow

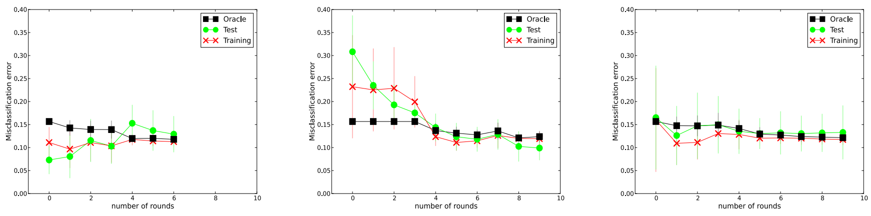


(d) Hybrid workflow—Effect of varying α

Fig. 7. The estimated training error, test error, and the actual error (oracle) for (a) the parallel dump, (b) the serial waterfall and (c) the hybrid workflow. (d) The effect of varying the split parameter α.

of sentences one can construct various features which quantify the dissimilarity between two sentences. One of the most import set of features are the based on machine translation (MT) metrics. For example the BLEU score [11] (which measures n-gram overlap) which is an widely used evaluation metric for MT systems is an important feature for paraphrase detection. In our system we used a total of 14 such features and then trained a binary decision tree using the labeled data to train the classifier.

To collect the labeled data we show a pair of sentences to three in-house annotators and ask them to label them as either semantically equivalent or not. The sentences were taken from wikipedia articles corresponding to a specified topic. Due to the overall project design the labeling proceeded one topic at a time. We currently have a labeled data of 715 sentence pairs from a total of 6 topics (56, 76, 88, 108, 140, 247 sentence for each of the six topics) of which 112 were annotated as semantically equivalent by a majority of the annotators. We analyse a situation where the data arrives one topic at a time. The new data is allocated into train, validation, and test splits according the parallel dump, serial waterfall, and the hybrid workflow. At each round a binary decision tree is trained using the train split, the decision tree parameters are chosen using the validation split, and the model performance is assessed using the test split.



(a) Parallel (b) Serial (c) Hybrid ($\alpha = 0.5$)

Fig. 8. *Paraphrase detection* (see § 7) (a) The misclassification error at each round (until all data has been used) on the oracle, train and test split for (a) the parallel dump (with parameters $\beta = 0.6$ and $\gamma = 0.2$), (b) the serial waterfall (with $\delta_1 = 0.5$, $\delta_2 = 0.3$, and $\delta_3 = 0.2$), and (b) the hybrid workflow (with $\alpha = 0.5$). The oracle is a surrogate for the true model performance which is evaluated on 30 % of entire original data.

Figure 8 shows the misclassification error on the train and the test splits as a function of the number of rounds, here each round refers to a point in time when we acquire a new labeled data and data reallocation/movement happens. The results are averaged over 50 replications, where for each replication the order of the topics is randomly permuted. We would like to see how close is the model performance assessed using the test split to the true model performance on the entire data (which we call the oracle). Since we do not have access to the true data distribution we sequester 30 % of original data (which includes data from all topics) and use this as a surrogate for the true performance. The following observations can be made (see Figure 8): (1) The test error for all the three workflows approaches the true oracle error at steady state when a large amount of data has been collected. (2) However in the early stages the performance of the model on the test split as assessed by the parallel workflow (Figure 8(a)) is relatively optimistic while that of the serial workflow(Figure 8(b)) is highly pessimistic. (3) The proposed hybrid workflow (see Figure 8(d)) estimates the test error much closer to the oracle error.

8 Related Work

There is not much related work in this area in the machine learning literature. Most earlier research has focussed on settings with either unlimited data or finite fixed data, while this paper proposes data flow strategies for finite but growing datasets. The bias due to repeated use of the test set has been pointed out in a few papers in the cross-validation setting [10,13]. However main focus of this paper is on data rich situations where we estimate the prediction capability of a classifier on a independent test data set (called the *conditional test error*). In data poor situations techniques like cross-validation [7] and bootstrapping [3] are widely used as a surrogate to this, but they can only estimate the *expected test error* (averaged over multiple training sets).

There is a rich literature in the area of learning under concept drift [5] and dataset shifts [12]. Concept drift primarily refers to an online supervised learning scenario when the relation between the input data and the target variable changes over time. Dataset shift is a common problem in predictive modeling that occurs when the joint distribution of inputs and outputs differs between training and test stages. Covariate shift, a particular case of dataset shift, occurs when only the input distribution changes. The other kinds of concept-drifts are *prior probability shift* where the distribution over true label y changes, *sample selection bias* where the data distributions varies over time because of an unknown sample rejection bias and *source component shift* where the datastream can be thought to be originating from different unknown sources at different time points.

Various strategies have been proposed to correct for these shifts in test distribution [5,12]. In general the field of concept drift seeks to develop shift-aware models that can capture these specific types of variations or a combination of the different modes of variations or do model selection to assess whether dataset shift is an issue in particular circumstances [9]. In our current work the focus is to investigate into different kinds of dynamic workflows for assigning data into train, test and validation splits to reduce the effect of bias in the scenario of a time-shifting dataset. In our setting the drift arises as a consequence of that data arriving in an non-iid fashion. Hence this body of work mainly deals with the source component shift of the datasets where for example in the medical domain where a classifier is built from the training data obtained from various hospitals, each hospital may have a different machine with different bias, each producing different ranges of values for the covariate x and possibly even the true label y.

We are mainly concerned with allocating the new data to the existing splits and not with modifying any particular learning method to account for the shifts. One of our motivations was not to make the allocation strategies model or data distribution specific. We wanted to come up with strategies that can be used with any model or data distribution. The main contribution of our work is to propose these strategies and empirically analyze them. These kind of strategies have not been discussed in the concept drift literature.

9 Conclusions

We analysed three workflows for allocating new data into the existing training, validation, and test splits. The parallel dump workflow splits the data into three parts and directly dumps them into the existing splits. While it can exploit the new data quickly to build a good model the estimate of the model performance is optimistic especially when the new data does not arrive in a random fashion. The serial waterfall workflow which trickles the data from one level to another avoids this problem by keeping the test set fresh and prevents the bias due to multiple reuse of the test set. However it takes a long time for the new data to reach the training split. The proposed hybrid workflow which balances both the workflows seems to be a good compromise—it can give a good model reasonably fast and at the same time produce a reasonably good estimate of the model performance.

A Appendix: Bias Due to Test Set Reuse

We will consider a scenario where we have a test set of N examples. At each reuse we supplement the test set with M new examples and remove M of the oldest examples. In this appendix we prove the following result:

After each reuse if we supplement the test set (consisting of N samples) with $M > N/(k\sqrt{N}+1)$ new samples for some small constant k then the bias due to test set reuse can be safely ignored.

Maximum Bias Due to Test Set Reuse: Let E_j be the estimated error of the model $\widehat{f}_j$ on a test set consisting of N samples after the test set having been reused j times, *i.e.*,

$$\mathrm{E}_j = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \widehat{y}_{ij}) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \widehat{f}_j(\boldsymbol{x}_i)), \tag{1}$$

where y_i is the true response, $\widehat{y}_{ij} = \widehat{f}_j(\boldsymbol{x}_i)$ is the response predicted by the learnt model after the test set has been reused j times, and L is the loss function used to measure the error. If the test set is used multiple times the final model will overfit to the testing set. In other words the performance of the model on the test set will be biased and will not reflect the true performance of the model.

Let $\mathrm{Bias}_{max}(\mathrm{E}_j)$ be the maximum possible bias in the estimate of the error E_j caused due to multiple reuse of the test set. The first time the test set is used the bias is zero, *i.e.*, $\mathrm{Bias}_{max}(\mathrm{E}_1) = 0$. Every subsequent use increases the bias. The worst case scenario (the perhaps the easiest for the developer) is when the developer directly observes the predictions $\widehat{y}_{ij}$ and learns a model on these predictions to match the desired response y_j for all examples in the test set. Since we have N examples in the test set by reusing the test set $N+1$ times one can actually drive the error $\mathrm{E}_{N+1} = 0$ since we will have N unknowns to estimate $(y_1, \ldots, y_N)$ and N new tests. Hence $\mathrm{Bias}_{max}(\mathrm{E}_{N+1}) = E_1$. We will further assume that at each test we lose one degree of freedom and hence approximate

$$\mathrm{Bias}_{max}(\mathrm{E}_j) = \frac{j-1}{N}E_1, \quad j = 1, \ldots, N. \tag{2}$$

Supplement the Test Set to Avoid the Bias: In order to avoid this bias our strategy is to supplement the test set with M new examples and move the oldest M examples to the lower validation set. We do this every time we reuse the test set. If we keep doing this then in the long run we will have a set of N examples which has been reused N/M times. Hence if we supplement the test set with M examples at each reuse the bias in the test set at steady state will be

$$\mathrm{Bias}_{max}(\mathrm{E}_\infty) = \frac{N-M}{NM}E_1, \tag{3}$$

where E_∞ is the error in this steady state scenario.

How Much to Supplement?: If we require that for some small constant k, $|\text{Bias}_{max}(\text{E}_\infty)| < \text{sd}(\text{E}_\infty)k$ then the bias can be safely ignored, where $\text{sd}(\text{E}_\infty)$ is the standard deviation of the error estimate. For analytical tractability we will assume a squared loss error function, *i.e.*, $L(y, \widehat{y}) = (y - \widehat{y})^2$. Hence at steady state

$$\text{E}_\infty = \frac{1}{N}\sum_{i=1}^{N}(y_i - \widehat{y}_{i\infty})^2. \tag{4}$$

Since we have a sum of squares we assume $\text{E}_\infty \sim \Gamma(N, \sigma^2/N)$, a gamma distribution with N degrees of freedom and $\sigma^2 = \text{Var}(y_i - \widehat{y}_{i\infty})$. Hence

$$\text{sd}(\text{E}_\infty) = \sigma^2/\sqrt{N}. \tag{5}$$

We also approximate E_1 by its expected value $E_1 \approx E[E_1] = \sigma^2$.

Hence $|\text{Bias}_{max}(\text{E}_\infty)| < \text{sd}(\text{E}_\infty)k$ implies $\frac{N-M}{NM}\sigma^2 < \frac{\sigma^2}{\sqrt{N}}k$ that is $M > \frac{N}{k\sqrt{N}+1}$. Hence after each reuse if we supplement the test set (consisting of N samples) with $M > N/(k\sqrt{N}+1)$ new samples for some small constant k then the bias due to test set reuse can be safely ignored. Note that M has approximately $\sqrt{N}$ dependency.

References

1. Kaggle. www.kaggle.com
2. Chatfield, C.: Model uncertainty, data mining and statistical inference. Journal of the Royal Statistical Society. Series A (Statistics in Society) **158**(3), 419–466 (1995)
3. Efron, B., Tibshirani, R.: An introduction to the bootstrap. Chapman and Hall (1993)
4. Faraway, J.: Data splitting strategies for reducing the effect of model selection on inference. Computing Science and Statistics **30**, 332–341 (1998)
5. Gama, J., Zliobaite, I., Bifet, A., Pechenizkiy, M., Bouchachia, A.: A survey on concept drift adaptation. ACM Computing Surveys **46**(4) (2014)
6. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer Series in Statistics (2009)
7. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. IJCAI **14**, 1137–1145 (1995)
8. Madnani, N., Tetreault, J., Chodorow, M.: Re-examining machine translation metrics for paraphrase identification. In: Proceedings of 2012 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL 2012), pp. 182–190 (2012)
9. Mohri, M., Rostamizadeh, A.: Stability bounds for non-i.i.d. processes. In: Advances in Neural Information Processing Systems, vol. 20, pp. 1025–1032 (2008)
10. Ng, A.Y.: Preventing overfitting of crossvalidation data. In: Proceedings of the 14th International Conference on Machine Learning, pp. 245–253 (1997)
11. Papineni, K., Roukos, S., Ward, T., Zhu, W.J.: BLEU: a method for automatic evaluation of machine translation. In: Proceedings of the 40th Annual meeting of the Association for Computational Linguistics (ACL-2102), pp. 311–318 (2002)

12. Quinonero-Candela, J., Sugiyama, M., Schwaighofer, A., Lawrence, N.D. (eds.): Dataset Shift in Machine Learning. Neural Information Processing series. MIT Press (2008)
13. Rao, R.B., Fung, G.: On the dangers of cross-validation. An experimental evaluation. In: Proceedings of the SIAM Conference on Data Mining, pp. 588–596 (2008)
14. Samuelson, F.: Supplementing a validation test sample. In: 2009 Joint Statistical Meetings (2009)
15. Stone, M.: Asymptotics for and against crossvalidation. Biometrika **64**, 29–35 (1977)