Notes on Single-Pass Online Learning Algorithms

Vitor R. Carvalho and William W. Cohen

CMU-LTI-06-002

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA
www.lti.cs.cmu.edu

July 6, 2006

Abstract

Online learning methods are typically faster and have a much smaller memory footprint than batch learning methods. However, in practice online learners frequently require multiple passes over the same training data in order to achieve accuracy comparable to batch learners. We investigate the problem of single-pass online learning, i.e., training only on a single pass over the data. We compare the performance of single-pass online learners to traditional batch learning, and we propose a new modification of the Margin Balanced Winnow algorithm that can reach results comparable to linear SVM for several NLP tasks. We also explore the effect of averaging, a.k.a. *voting*, on online classifiers. We provide experimental evidence that voting can be successfully used to boost the performance of several single-pass online learning algorithms. Finally, we describe how the Modified Margin Balanced Winnow algorithm proposed can be naturally adapted to perform online feature selection. This scheme performs comparably to information gain or chi-square, with the advantage of being able to select features on-the-fly.

Contents

1 Introduction 4

2 Online Learning Methods 5

2.1 Perceptron, ROMMA and Passive-Aggressive 5

2.2 Winnow-based Learners 6

2.2.1 Positive Winnow 7

2.2.2 Balanced Winnow 7

2.2.3 Modified Balanced Winnow 7

3 Averaging (a.k.a. Voting) 8

4 Experiments 9

4.1 Datasets 9

4.2 Parameters and Baselines 9

4.3 Results 10

4.3.1 Experiments 10

4.3.2 Effect of Voting 11

4.3.3 MBW Learning Curves 12

5 Online Feature Selection 13

5.1 Introduction 13

5.2 Extremal Feature Selection 14

5.3 Comparative Results 15

6 Related Work 15

7 Conclusions 16

1 Introduction

There are several reasons to use online learning methods. When compared to batch learning, online methods are usually faster, simpler to implement and have a much smaller memory footprint. Online learning methods have been the subject of research for many years [12]. The Winnow algorithm, for instance, was proposed in the 1980s [20]. Some initial experimental work on Winnow was performed by Dagan et al.[11] and Blum [5], suggesting that Winnow-based learners can be very successful in tasks such as calendar scheduling and text categorization. In fact, some of the appealing theoretical properties of Winnow include the ability to effectively handle domains with high dimensionality and sparse data [12].

More recently, a different implementation of Winnow has proven successful in the task of email folder classification [3]. By using a "wide-margin", considering all features binary and training Winnow using several passes through the data, Bekkerman et al. reached performance numbers comparable to standard batch leaning algorithms such as SVM or Logistic Regression.

Online learning algorithms have been traditionally trained using several passes through the training data [3, 11, 16]. In the current work we address the problem of single-pass online learning, i.e., online learning restricted to a single training pass over the available data. This setting is particularly relevant when the system cannot afford several passes throughout the training set: for instance, when dealing with massive amounts of data, or when memory or processing resources are restricted, or when data is not stored but presented in a stream.

Here we experimentally compare the performance of different online learners (ROMMA, Perceptron, Passive-Aggressive, Winnow, Balanced Winnow, etc) to traditional batch learning in the single-pass setting, and we introduce a new online algorithm — MBW or Modified Balance Winnow — that outperforms all other single-pass online learners and achieves results comparable to Linear SVM in several NLP tasks.

Voting (a.k.a. averaging) an online classifier is a technique that, instead of using the best hypothesis learned so far, uses a weighted average of all hypotheses learned during a training procedure. The averaging procedure is expected to produce more stable models, which leads to less overfitting [14]. Averaging techniques have been successfully used on the Perceptron algorithm [16], but never in other online learners such as Winnow, Passive-Aggressive[10] or ROMMA[19]. In the current work, we provide a detailed performance comparison on how averaging affects the aforementioned online learners when restricted to a single learning pass only. Results clearly indicate that voting improves performance of most mistake-driven learning algorithm, including learners to which it has not traditionally been applied.

Feature selection techniques provide an efficient way to work with datasets that have a large number of features, or in settings with limited memory resources. Selecting the "most meaningful" features from a large dataset can reduce training and testing times, facilitate data visualization and understanding, reduce memory requirements and, in some cases, improve prediction accuracy. There are several available techniques to do feature selection in NLP tasks, but extending such feature selection techniques to the online learning setting is not straightforward. In the online setting, the entire feature set is not known in advance, and the goal is to refine the model every time new examples are presented to the learner. Not only can new features be added to the model at each time step, but also features previously selected can be discarded from the model. Batch techniques for feature selection are known, but are expensive to apply in an online setting.

In this work, we propose a very simple and fast Online Feature Selection (OFS) scheme based on the “extreme” weights stored by MBW. Performance results clearly indicate that this scheme can be effectively applied to NLP problems, being competitive with Chi-Square or Information Gain, but having the advantage of selecting the best features on-the-fly.

This report is organized as follows. Section 2 presents different mistake-driven online learners and introduce the MBW algorithm. Section 3 presents the voting technique as a generic scheme for online learning. Section 4 compares the learner’s results, and presents the first two contributions: the impressive results of MBW in NLP tasks and the boost in performance by averaging classifiers in non-NLP datasets. The last contribution of the paper is presented in Section 5, where we introduce a new online feature selection scheme based on the MBW learner and demonstrate its effectiveness. Finally, Sections 6 and 7 present the related work and our conclusions.

2 Online Learning Methods

There are several reasons to use online learning methods. When compared to batch learning, online methods are usually faster, simpler to implement and have a much smaller memory footprint. For such reasons, these techniques scale up very well to very large scale systems. Additionally, they can easily be converted to batch algorithms.

The general format of mistake-driven online learning algorithms is described in Figure 1. For each new example x_t presented, the current model will make a prediction $\widehat{y_t} \in \{-1, 1\}$ (based on the score function f, the example x_t and the current weight vector w_i), and compare it to the true class $y_t \in \{-1, 1\}$. In the case of a prediction mistake, the model will be updated. Different mistake-driven algorithms will differ in terms of the score function f and in the way the weights w_i are updated, as we shall detail in the next sections.

Table 1: Mistake-Driven Online Classifier.

```
1. Initialize $i = 0$, success counter $c_i = 0$, initial model $w_0$
2. For $t = 1, 2, ..., T$ do:
   (a) Receive new example $x_t$
   (b) Make prediction $\widehat{y_t} = f(w_i, x_t)$ and receive true class $y_t$
   (c) Compare $\widehat{y_t}$ to $y_t$. If prediction was mistaken:
       i. Update model $w_i \rightarrow w_{i+1}$
       ii. $i = i + 1$
   (d) Else
       i. $c_i = c_i + 1$
```

2.1 Perceptron, ROMMA and Passive-Aggressive

Initially proposed in 1958 [25], the Perceptron algorithm uses a very simple and effective update rule. The initial model w_0 is usually a vector with zero weights only. Its decision function is given by $f = sign(\langle x_t, w_i \rangle) = sign(\sum_j w_i^j x_t^j)$, where x_t^j is the weight of the input element j in the incoming example x_t and w_i^j is the weight of element j in w_i. The model is improved based on additive updates, i.e., $w_{i+1} = w_i + x_t \cdot y_t$. In spite of its simplicity, given a linearly separable training set, the Perceptron algorithm is guaranteed to find a solution that perfectly classifies this training set in a finite number of iterations.

The Passive-Aggressive algorithm [10] is also based on additive updates of the model weights. However, the update policy here is based on an optimization problem closely related to the one solved in Support Vector Machine techniques. The initial model w_0 is usually a vector with zeros only. Used in classification mode. Its decision function is given by $f = sign(\langle x_t, w_i \rangle)$, and the update rule is $w_{i+1} = w_i + x_t \cdot y_t \cdot \tau_t$. The scaling factor τ_t in non-realizable cases should follow:

$$\tau_t = \frac{Loss(x_t, w_i)}{\|x_t \cdot y_t\|^2 + \gamma} = \frac{\epsilon - y_t\,(x_t \cdot w_i)}{\|x_t \cdot y_t\|^2 + \gamma} \tag{1}$$

where the relaxation parameter $\gamma \geq 0$, and ϵ is the insensitivity parameter.

The Relaxed Online Maximum Margin Algorithm, or ROMMA [19], incrementally learns linear threshold functions. It attempts to find a function that classifies previous examples correctly with a maximum margin. ROMMA uses additive as well as multiplicative updates to the model weights. The initial model w_0 is set as a vector of zero weights only. When this classifier is used in non-aggressive mode, its decision function is given by $f = sign(\langle x_t, w_i \rangle)$, and the update rule is $w_{i+1} = c_t \cdot w_i + x_t \cdot d_t$; where the multiplicative and additive update coefficients are given by:

$$c_t = \frac{\|x_t\|^2 \|w_i\|^2 - y_t\,(x_t \cdot w_i)}{\|x_t\|^2 \|w_i\|^2 - (x_t \cdot w_i)^2} \tag{2}$$

$$d_t = \frac{\|w_i\|^2\,(y_t - (x_t \cdot w_i))}{\|x_t\|^2 \|w_i\|^2 - (x_t \cdot w_i)^2} \tag{3}$$

2.2 Winnow-based Learners

We present three learning algorithms based on the Winnow concept: Positive Winnow, Balanced Winnow and the Modified Balanced Winnow. For all three, we assume the incoming example x_t is a vector of positive weights, i.e., $x_t^j \geq 0$, $\forall t$ and $\forall j$. This assumption is usually satisfied in NLP tasks, where the x_t^j values are typically the frequency of a term, presence of a feature, TFIDF value of a term, etc.

In preliminary experiments, we found that the Winnow variants performed better if we applied an *augmentation* and a *normalization* preprocessing step, in both learning and testing phases. When learning, the algorithm receives a new example x_t with m features, and it initially *augments* the example with an additional feature(the $(m+1)^{th}$ feature), whose value is permanently set to 1. After *augmentation*, the algorithm then *normalizes* the sum of the weights of the augmented example to 1, therefore restricting all feature weights to $0 \leq x_t^j \leq 1$.

In testing mode, the *augmentation* step is the same, but there is a small modification in the *normalization*. Before the normalization of the incoming instance, the algorithm checks each feature in the instance to see if it is already present in the current model (w_i). The features not present in the current model are then removed from the incoming instance before the normalization takes place.

2.2.1 Positive Winnow

Positive Winnow[20] is a popular Winnow-based algorithm with 3 parameters: a promotion parameter $\alpha > 1$, a demotion parameter β, where $0 < \beta < 1$, and a threshold parameter $\theta_{th} > 0$.

Let $\langle x_t, w_i \rangle$ denote the inner product of vectors x_t and w_i. Positive Winnow uses a decision function given by $f = sign(\langle x_t, w_i \rangle - \theta_{th})$, and the update rule is: For all j s.t. $x_t^j > 0$,

$$w_{i+1}^j = \begin{cases} w_i^j \cdot \alpha & \text{, if } y_t > 0 \\ w_i^j \cdot \beta & \text{, if } y_t < 0 \end{cases}$$

2.2.2 Balanced Winnow

Another version of Winnow that has been successfully used in the literature is Balanced Winnow. Here, the model w_t is a combination of two parts: a positive model u_t and a negative model v_t. Balanced Winnow also has a promotion parameter α, a demotion parameter β and threshold θ_{th}, which follow the same range restrictions previously seen in Positive Winnow.

The score function is $f = sign(\langle x_t, u_i \rangle - \langle x_t, v_i \rangle - \theta_{th})$, and the update rule follows from:

$$\text{For all } j \text{ in } x_t\text{:} \quad u_{i+1}^j = \begin{cases} u_i^j \cdot \alpha & \text{, if } y_t > 0 \\ u_i^j \cdot \beta & \text{, if } y_t < 0 \end{cases} \quad \text{and} \quad v_{i+1}^j = \begin{cases} v_i^j \cdot \beta & \text{, if } y_t > 0 \\ v_i^j \cdot \alpha & \text{, if } y_t < 0 \end{cases}$$

The initial model u_0 and v_0 are respectively set to the positive values θ_0^+ and θ_0^- in all dimensions.

Despite their simplicity, Positive Winnow and Balanced Winnow are able to perform very well in different NLP tasks [3, 5, 11]

2.2.3 Modified Balanced Winnow

We introduce here a modified version of the Balanced Winnow algorithm. The *Modified Balanced Winnow* algorithm, henceforth MBW, is detailed in Table 2.

Similar to Balanced Winnow, MBW has a promotion parameter α, a demotion parameter β, a threshold parameter θ_{th} and it also uses the same decision function as Balanced Winnow, i.e., $f = sign(\langle x_t, u_i \rangle - \langle x_t, v_i \rangle - \theta_{th})$. The initialization is the same: u_0 and v_0 are respectively set to the positive values θ_0^+ and θ_0^- in all dimensions. However, there are two modifications.

The first modification is the "thick"-separator (or wide-margin) idea used in [11]. The prediction is considered mistaken, not only when y_t is different from $\widehat{y_t}$, but also when the score function multiplied by y_t is smaller than the "margin" M, where $M \geq 0$. More specifically, the mistake condition is $(y_t \cdot (\langle x_t, u_i \rangle - \langle x_t, v_i \rangle - \theta_{th})) \leq M$.

The second modification is a small change in the update rules, such that each multiplicative correction will depend on the particular feature weight of the incoming example. The change is illustrated in Table 2.

The second modification is a small change in the update rules in order to make it more aggressive. The MBW update rules for all j in x_t are:

$$u_{i+1}^j = \begin{cases} u_i^j \cdot \alpha \cdot (1 + x_t^j) & \text{, if } y_t > 0 \\ u_i^j \cdot \beta \cdot (1 - x_t^j) & \text{, if } y_t < 0 \end{cases} \quad \text{and} \quad v_{i+1}^j = \begin{cases} v_i^j \cdot \beta \cdot (1 - x_t^j) & \text{, if } y_t > 0 \\ v_i^j \cdot \alpha \cdot (1 + x_t^j) & \text{, if } y_t < 0 \end{cases}$$

Table 2: Modified Balanced Winnow.

```
1. Initialize $i = 0$, success counter $c_i = 0$, initialize models $u_0$ and $v_0$
2. For $t = 1, 2, ..., T$ do:
    (a) Receive new example $x_t$, and augment it with "always-on" feature.
    (b) Normalize $x_t$ to 1.
    (c) Calculate $score = \langle x_t, u_i \rangle - \langle x_t, v_i \rangle - \theta_{th}$.
    (d) Receive true class $y_t$.
    (e) If prediction was mistaken, i.e., $(score \cdot y_t) \leq M$:
        i. Update models. For all $j$ in $x_t$:
           $$u_{i+1}^j = \begin{cases} u_i^j \cdot \alpha \cdot (1 + x_t^j) & \text{, if } y_t > 0 \\ u_i^j \cdot \beta \cdot (1 - x_t^j) & \text{, if } y_t < 0 \end{cases}$$
           $$v_{i+1}^j = \begin{cases} v_i^j \cdot \beta \cdot (1 - x_t^j) & \text{, if } y_t > 0 \\ v_i^j \cdot \alpha \cdot (1 + x_t^j) & \text{, if } y_t < 0 \end{cases}$$
        ii. $i = i + 1$
    (f) Else
        i. $c_i = c_i + 1$
```

3 Averaging (a.k.a. Voting)

Averaging a classifier is a technique that, instead of using the best hypothesis learned so far, uses a weighted average of all hypotheses learned during the training procedure. The averaging procedure is expected to produce more stable models, which leads to less overfitting [14].

An averaged version of the Perceptron learner (a.k.a. Voted Perceptron or v-P) is described in [15]. In this work the final hypothesis of the Perceptron learner is an average of the intermediary hypotheses weighted by the number of correct predictions that each hypothesis made in the learning process. More specifically, referring to Table 1, the averaged model w_a would be expressed as:

$$w_a = \frac{1}{Z} \sum_i w_i \cdot c_i$$

, where c_i is the number of correct predictions made by the intermediary hypothesis w_i, and $Z = \sum_i c_i$ is the total number of correct predictions made during training.

Averaging can be trivially applied to any mistake-driven online algorithm. In the current work we apply it to all learners presented previously and we refer to it using a "v-" prefix (from Voted). For instance, v-Winnow and v-ROMMA refer to voted (or averaged) versions of Winnow and ROMMA, respectively.

4 Experiments

4.1 Datasets

The algorithms described above were evaluated in several datasets, from different sources. The *RequestAct* dataset [8] labels email messages as having a "Request speech act" or not. In addition to the single word features [8], all word sequences with a length of 2, 3, 4 or 5 tokens were extracted and considered to be different features. The *RequestAct* dataset has 70147 features and 518 examples. The *Spam* dataset has 3302 examples and 118175 features. The task is to detect spam email messages [1]. The *Scam* dataset has 3836 examples and 121205 features. Here we attempt to separate "Scam" messages from the others [1]. The *Reuters* dataset [18] has 11367 examples and 30765 features. We attempt to classify the category "money" [2]. The *20newsgroup* dataset [21, 22] has 5000 examples and 43468 features, and the problem is classifying newsgroups posts according to one of the topics. The *MovieReviews* dataset [23] has 1400 examples and 34944 features. In this problem we try to associate a positive or negative sentiment with a movie review. The *Webmaster* dataset has 582 examples and 1406 features. The task is to classify web site update requests as "Change" from "Add or Delete" [9].

The *Signature* and the *ReplyTo* datasets are related to the tasks of detecting signature lines and "reply-to" lines in email messages, respectively, using a basic set of features [6]. Both datasets have a total of 37 features and 33013 examples.

The next datasets were obtained from the UCI data repository. The *Adult* dataset originally had 14 attributes and, using only the training partition provided, 30162 examples. Examples with missing attributes were discarded and the 8 nominal attributes were turned into different binary attributes. The final dataset had 104 different attributes. The *Congressional* dataset has 16 binary features and has 435 examples. The *Credit* (or Japanese Credit Screening) dataset has 690 examples and 15 features originally. After removing examples with missing attributes and turning nominal attributes into different binary features, the dataset had 46 features and 653 examples. The *Ads* dataset (or Internet Advertisements) has 3279 examples and 1558 features, mostly binary. Missing features were disregarded in this data. The *WiscBreast* database represents the breast cancer database obtained from the University of Wisconsin. The data has 9 integer-valued features, and after removing examples with missing attributes, a total of 683 examples and 89 features remained. The *Nursery* dataset has 12960 instances and originally 8 nominal features. After turning nominal features into different binary features, 89 features can be found in the dataset. The task here is to distinguish between "priority" and the other classes.

4.2 Parameters and Baselines

We used an implementation of Passive-Aggressive with parameters $\epsilon = 1$ and $\gamma = 0.1$; these values were arbitrarily chosen based on preliminary tests.

In all Winnow variants, we set the promotion parameter $\alpha = 1.5$ and the demotion parameter $\beta = 0.5$. These are the same values used in previous Winnow implementations [3, 5]. Additionally, all Winnow variants used threshold $\theta_{th} = 1.0$ motivated by the fact that all incoming examples go through the *normalization* preprocessing step. Also motivated by the *normalization* procedure, the "margin" M was set to 1.0 in MBW.

Based on ideas from Dagan et. al. [11], the initial weights in Positive Winnow were initialized as $\theta_0 = 1.0$, and in Balanced Winnow as $\theta_0^+ = 2.0$ and $\theta_0^- = 1.0$. Similar to Balanced Winnow, the MBW initial weights were $\theta_0^+ = 2.0$ and $\theta_0^- = 1.0$. Table 3 summarizes the parameters values.

For comparison, we added performance results of two popular learning algorithms that are typically used in batch mode: linear *SVM*[1](Support Vector Machine) and *Naive Bayes* [21]. Results were evaluated in terms of Error Rate or F1 measures. F1 is the harmonic precision-recall mean, defined as $F1 = \frac{2 \cdot Precision \cdot Recall}{Recall + Precision}$.

Learner	Parameters
Passive-Aggressive	$\epsilon = 1$ and $\gamma = 0.1$
Winnow	$\alpha = 1.5$, $\beta = 0.5$, $\theta_{th} = 1.0$, $\theta_0 = 1.0$
Balanced Winnow	$\alpha = 1.5$, $\beta = 0.5$, $\theta_{th} = 1.0$, $\theta_0^+ = 2.0$, $\theta_0^+ = 1.0$
MBW	$\alpha = 1.5$, $\beta = 0.5$, $\theta_{th} = 1.0$, $\theta_0^+ = 2.0$, $\theta_0^+ = 1.0$, $M = 1.0$

Table 3: Parameters of Learners

4.3 Results

4.3.1 Experiments

Initially we evaluated the general classification performance of the algorithms in 5-fold cross-validation experiments. All algorithms were trained using only a single pass through the training data. Results are illustrated in Tables 4 and 5.

Table 4 describes the performance of five different online methods, along with their voted versions. The first eight datasets in Table 4 are NLP-like datasets, where the feature space is very large and the examples are typically "sparse", i.e., the number of non-zero features in the examples is much smaller than the size of the feature space. The last seven datasets (non-NLP) in Table 4 have a much smaller feature space and the examples are not sparse.

Median F1 values and the average rank values over the two different types of data are also included in Table 4. The best results for each dataset are indicated in bold. Two-tailed T-Tests relative to MBW are indicated with the symbols * ($p \leq 0.05$) or ** ($p \leq 0.01$).

In general, the non-voted Winnow variants performed better (higher Median F1 and lower Avg. Rank) than non-voted Passive-Aggressive or non-voted ROMMA on both types of datasets. Passive-Aggressive typically presented better results than ROMMA; and Balanced Winnow outperformed Positive Winnow in almost all tests. MBW outperformed all other online learners for NLP datasets, and also all other non-voted learners for non-NLP datasets.

We compare MBW results to the batch learners SVM and Naive Bayes in Table 5. This Table illustrates F1 results along with their standard errors for SVM, Voted Perceptron (or v-P), MBW, v-MBW and Naive Bayes learners. Similar to Table 4, the datasets are presented in two groups (NLP and non-NLP) and best results are indicated in bold.

From Tables 4 and 5, it is important to observe that the MBW learner indeed reaches impressive performance numbers in the NLP-like datasets, outperforming all other learners — including SVM. The MBW performance in the NLP dataset is very encouraging, and a more detailed analysis of the behavior of this learner on NLP datasets will be presented in Section 4.3.3.

[1]We used the LIBSVM implementation [7] with default parameters

NLP Datasets	MBW	PW	BW	PA	ROMMA	v-MBW	v-PW	v-BW	v-PA	v-ROMMA
RequestAct	**76.7**	67.0**	62.6**	68.9*	09.6**	67.3**	46.8**	59.0**	60.2**	5.6**
Spam	95.8	93.8**	94.4	93.1**	83.1**	95.8	94.0**	**96.2**	93.3**	73.3**
Scam	**99.9**	96.5**	98.4**	99.2**	97.3**	99.8	98.4**	99.6	97.6**	95.6**
Reuters	95.9	93.8**	94.0**	95.5	91.9	**96.9****	95.8	96.2	96.3	90.4**
20newsgroup	**93.7**	81.6**	86.6**	81.1**	66.9**	91.9	82.7**	87.3**	73.9**	53.7**
MovieReviews	75.1	66.8**	74.5	28.8**	57.1**	**77.2**	63.0**	68.9**	67.5**	24.8**
Webmaster	**88.6**	82.5	85.6	82.5	79.1**	86.7	82.0*	86.8	86.7	63.8**
Ads	**81.3**	73.8*	72.7*	70.0**	19.7**	78.2	71.7**	72.2**	63.6**	17.2**
Median F1	**91.1**	82.0	86.1	81.8	73.0	89.3	82.3	87.0	80.3	58.8
Avg. Rank	**1.75**	6.12	4.62	6.12	8.75	3.71	6.25	3.50	5.75	10.0
nonNLP Data.										
Sig	80.2	66.4**	74.1*	67.0*	60.9**	**80.3**	80.2	**80.3**	79.6	79.6
Reply	93.4	89.9	93.2	92.0	90.0	93.5	93.6	93.6	**94.2**	**94.2**
Adult	25.0	46.7**	44.7**	13.4**	41.8**	19.6**	**49.8****	49.1**	18.8**	41.0**
Congress	94.2	92.5*	93.6	92.4	93.3*	**96.0**	94.3	95.2	94.3	92.5
Credit	72.1	79.1	74.3	46.2**	59.3**	**79.7**	78.1	77.3	60.0**	66.9
Wisc	96.8	96.4	96.3	**97.5**	96.0	97.2	96.9	96.7	97.4	95.7
Nursery	69.6	55.8*	69.1	72.0	68.3	69.6	80.3**	83.1**	**86.3****	85.8**
Median F1	80.2	79.1	74.3	72.0	68.3	80.3	80.3	83.1	**86.3**	85.8
Avg. Rank	5.57	7.00	6.42	7.42	8.28	3.71	**3.14**	**3.14**	4.28	5.71

Table 4: General Performance of Single-Pass Online Learners – F1 measures (%). PW=Positive Winnow, BW=Balanced Winnow, PA=Passive-Aggressive. The symbols * and ** indicate paired t-Test statistical significance (relative to MBW) with $p \leq 0.05$ and $p \leq 0.01$ levels, respectively.

In the non-NLP tasks, however, SVM shows much better results than all other learners and MBW is not competitive at all. In fact, the Voted Perceptron would probably be the best choice for single-pass online learning in this type of data. It is interesting that the Voted Perceptron performs so well in non-NLP tasks and so poorly in NLP-like datasets.

In general, non-voted Winnow variants perform better in NLP-like than in non-NLP datasets. This agrees with the general intuition that multiplicative updates algorithms handle well high dimensional problems with sparse target weight vector (final hypothesis has many weights close to zero)[11].

4.3.2 Effect of Voting

Tables 4 and 5 present the overall effect of voting over the two types of datasets. These results can be visualized in Figure 1. This figure shows the extent to which voting can improve different online learners, and in which specific dataset types. In all four graphs, the F1 values of voted learners are placed in the y-axis, while the F1 measure of the non-voted learners are displayed in the x-axis.

From Figure 1 we can see that voting improves the performance of all online learners for non-NLP datasets: most points are above the (x=y) line. However, for the NLP-like datasets, the improvements due to averaging are not as obvious. For instance, on Balanced Winnow, a small improvement can be observed, particularly when the F1 values are high. On Positive Winnow and Passive-Aggressive, it is not clear if voting is beneficial. For ROMMA, voting visibly deteriorates the performance. For MBW, voting seems to causes a small performance deterioration.

In summary, voting seems to be a consistent and powerful way to boost the overall performance

	SVM	v-Perceptron	MBW	v-MBW	Naive Bayes
RequestAct	68.0 ± 1.5	65.4 ± 2.9	**76.7** ± 2.5	67.3 ± 2.2	56.85 ± 2.67
Spam	96.7 ± 1.1	69.0 ± 3.3	95.7 ± 0.4	95.7 ± 1.8	**97.4** ± 0.3
Scam	99.0 ± 0.3	94.2 ± 1.8	**99.9** ± 0.08	99.8 ± 0.1	99.62 ± 0.21
Reuters	96.7 ± 0.07	96.3 ± 0.17	95.9 ± 0.21	**96.8** ± 0.48	85.52 ± 0.54
20newsgroup	88.8 ± 0.39	67.9 ± 2.8	93.7 ± 0.4	91.9 ± 0.8	**94.42** ± 0.77
MovieReviews	**78.5** ± 1.26	71.4 ± 0.65	75.1 ± 0.9	77.1 ± 1.07	71.85 ± 0.71
Webmaster	**88.9** ± 2.5	88.5 ± 1.69	88.6 ± 1.9	86.6 ± 2.12	77.38 ± 4.51
Ads	80.5 ± 1.28	58.0 ± 10.1	**81.3** ± 1.2	78.2 ± 1.69	52.5 ± 0.72
Median F1	88.8	70.2	**91.1**	89.3	81.45
Avg. Rank	2.25	4.25	**2.12**	2.62	3.62
Signature	**80.3** ± 1.2	80.2 ± 1.58	80.2 ± 1.4	**80.3** ± 1.32	73.88 ± 1.37
Reply-to	**94.8** ± 0.8	94.3 ± 0.95	93.4 ± 0.7	93.5 ± 1.23	93.98 ± 0.83
Adult	32.3 ± 0.7	26.6 ± 0.9	25.0 ± 1.1	19.6 ± 0.8	**41.0** ± 6.7
Congressional	**96.2** ± 1.0	95.7 ± 0.86	94.2 ± 0.5	95.9 ± 1.19	91.7 ± 1.83
Credit	**80.2** ± 5.0	59.5 ± 1.79	72.1 ± 4.5	79.6 ± 2.69	66.78 ± 3.10
WiscBreast	96.6 ± 0.89	97.1 ± 0.39	96.8 ± 0.5	97.2 ± 0.4	**98.2** ± 0.3
Nursery	**87.1** ± 0.2	86.8 ± 0.56	57.0 ± 0.6	69.6 ± 0.4	84.4 ± 0.3
Median F1	**87.1**	86.8	80.2	80.3	84.4
Avg. Rank	**1.71**	3.00	4.00	3.00	3.14

Table 5: General Performance - F1 measure (%)

of very distinct single-pass online classifiers for non-NLP tasks. In NLP tasks, voting does not seem to bring the same benefits.

4.3.3 MBW Learning Curves

In previous sections we showed that single-pass MBW obtains surprisingly good results for different NLP tasks. Here we take a closer look in the learning rate of these algorithms, presenting a more detailed comparison with linear SVM and Voted Perceptron.

Figures 2 and 3 present, respectively, different test set F1 and test error rate learning curves for all eight NLP datasets. Each one of the figures illustrate the performance of MBW, Voted-MBW, Voted Perceptron (v-P) and SVM as a function of the number of examples presented to the learner. To mimic an online behavior from SVM, it was trained when the first 10 examples were presented and, after that, retrained anytime the training set size increased in 10%. A random split of 1/5 of the data was used as the test set, and the training set was presented to the learner sequentially. Similarly, Figure 4 illustrate the training error, i.e., the number of mistakes committed by the algorithm during the learning process.

Figures 2, 3 and 4 clearly indicate that the Voted Perceptron has typically the lowest performance, usually requiring more training examples than the other learners to reach the same performance levels. MBW presents impressive performance in most curves. Due to its aggressive updates, MBW tends to show more peaks in performance than Voted-MBW, particularly in early learning. Voted-MBW curves are smoother because it compensates the aggressive MBW updates with the averaging scheme. In general, the performances of MBW and Voted-MBW are much closer to SVM's performance than to Voted Perceptron's.

By combining excellent single-pass performance with the usual advantages of online learners, MBW presents several desirable characteristics for NLP tasks. In addition to that, MBW can

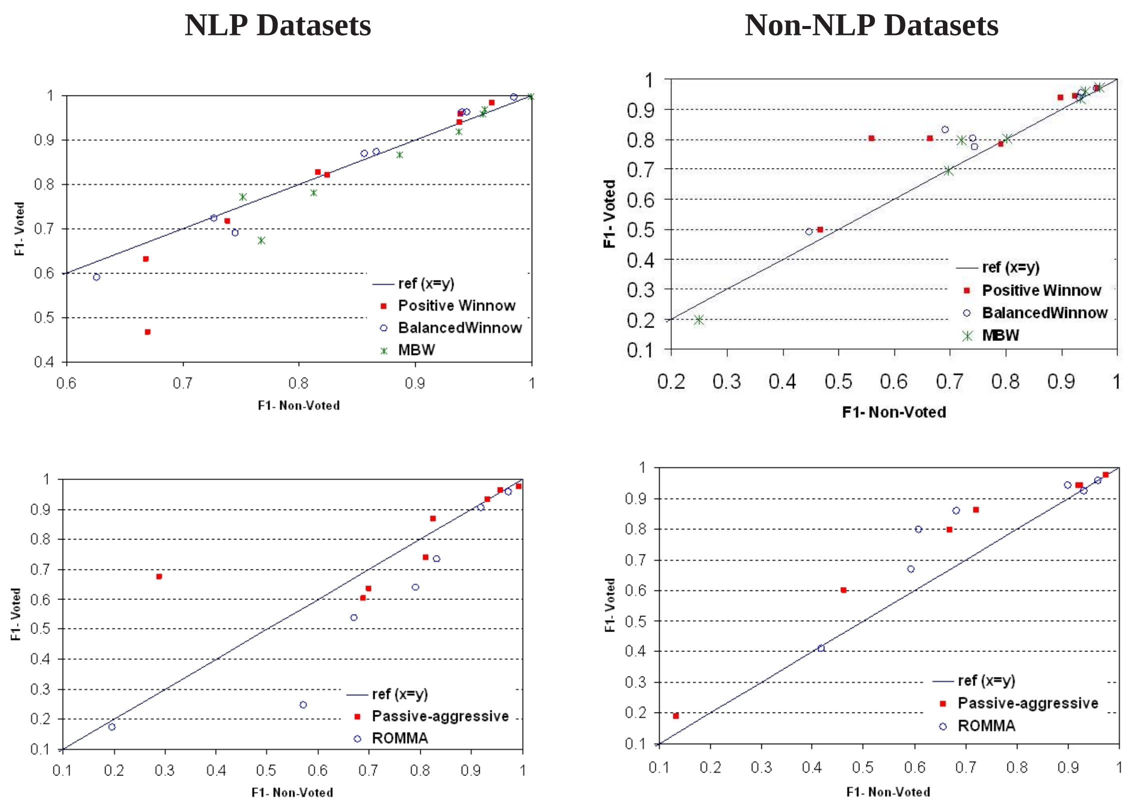


Figure 1: Effect of Voting Online learners

naturally be adapted to do effective feature selection in the online setting, as we will see in Section 5.

5 Online Feature Selection

5.1 Introduction

Feature Selection techniques provide an efficient way to work with datasets having a large number of features, or in settings with limited memory resources. Selecting the most "meaningful" features from a large dataset can reduce training and testing times, facilitate data visualization and understanding, reduce memory requirements and, in some cases, improve prediction accuracy. It can also be important for NLP tasks where the feature set is changing, and it is necessary to keep the size of classifiers from growing without bound.

Feature selection for NLP tasks is usually performed in batch mode. Given a dataset, all features are ranked according to some estimate of their "strength" in this particular task, and the selection is based on such a ranking. Examples of common metrics for feature selection are Information Gain and Chi-Square [13, 26].

Extending such batch feature selection techniques to the online learning setting is not obvious. In the online setting the complete feature set is not known in advance. Also, it would be desirable to refine the model every time new examples are presented to the learner: not only by adding new meaningful features to the model, but also by deleting unimportant features that were previously selected. Adapting the batch techniques to the online setting would be very expensive, since each score of each feature would need to be recalculated after every new example.

5.2 Extremal Feature Selection

As previously seen, the MBW learner reaches very good performance in NLP tasks with a single learning pass through the training data. Here we propose a very simple and very fast Online Feature Selection scheme called *Extremal Feature Selection* (or EFS), based on the weights stored by the MBW learner.

The idea is to rank the feature importance according to the difference (in absolute value) between its positive and negative MBW weights. More specifically, at each time t the importance score I of the feature j is given by

$$I_t^j = \left|u_t^j - v_t^j\right|$$

where u_t^j and v_t^j are the positive and the negative model weights for feature j.

After the scores I_t^j are computed, it would be expected that the largest values correspond to the most meaningful features to be selected at each time step. We would also expect that the lowest values of I_t^j correspond to the most unimportant features, prone to be deleted from the final model. In fact, a detailed analysis in the 20newsgroup dataset indicated that these low score features are typical *stop words*: "as, you, what, they, are, do, was, be, no, have, to, not".

In preliminary experiments, however, we observed an unexpected effect: performance is improved by selecting not only the highest I_t^j-valued features, but also a small number of features with the lowest values. Thus, as illustrated in Figure 5, EFS uses not only the extreme top T features, but also a small number from the extreme bottom B. For instance, in order to select 100 features from a dataset, EFS would select 90% of these 100 features from the extreme top T and 10% from the extreme bottom B.

The effectiveness of this idea can be seen in Figure 6. This figure illustrates the MBW test error rate for different numbers of features selected in the training set. More specifically, we first trained a MBW learner on the training set and then selected the features according to I_t^j-values and P. We then deleted the other features (non-selected) from the MBW model and used this final model as test probe. We used a random split of 20% of the 20newsgroup data as the test set, and the remaining as training examples. The quotient $P = \frac{T}{T+B}$ represents the fraction of the selected features with high-score I_t^j. For example, $P = 0.7$ indicates that 70% of the features were selected from the top and 30% from the bottom; and $P = 1$ indicates that no low-score features were selected at all.

As expected, the performances in Figure 6 improve as P increases — since selecting more high score features translates to better selection overall. This general behavior was also observed in all other NLP datasets. However, the condition $P = 0.9$ seems to outperform the condition $P = 1$, specially for non-aggressive feature selection (i.e., when the number of selected features is relatively large). This unexpected behavior was also observed in most NLP datasets (see also Figure 7). It indicates that there is a small set of low score features that are very effective to the proposed online feature selection scheme.

We speculate that the importance of these low score features is related to a smoothing-like effect in the MBW learner. Recall that MBW uses a *normalization* preprocessing step that is susceptible to the number of non-zero features in the incoming example, and the low-score features are frequently found in most examples.

5.3 Comparative Results

Similar to Figure 6, Figure 7 and 8 show test error rates in other datasets for different numbers of features selected in the training set. Again, 20% of the data was used as the test set and the quotient $P = \frac{T}{K}$ represents the fraction of the selected features with high-score I_t^j. Figure 7 also illustrates MBW performance using two popular batch feature selection schemes: Information Gain (IG) and Chi-Square(CHI).

Figures 7 and 8 reveal that EFS with $P = 0.9$ has a performance comparable to IG and CHI. In one of the experiments (MovieReviews) the EFS performance was generally better than IG or CHI in almost all feature selection ranges. Reuters was the only dataset where the traditional methods outperformed EFS, but only for very aggressive feature selection — when only a small number of features is selected.

EFS results are very encouraging. The ability to perform effective online learning and feature selection in the same framework can largely benefit systems constrained by limited resources.

6 Related Work

Online learning methods have been the subject of research for many years [12]. The Winnow algorithm, for instance, was proposed in the 1980s [20]. Some of the appealing theoretical properties of Winnow include the ability to effectively handle domains with high dimensionality and sparse data [12]. Some initial experimental work on Winnow was performed by Dagan et al.[11] and Blum [5], suggesting that Winnow-based learners can be very successful in tasks such as calendar scheduling and text categorization. In another task, a different implementation of Winnow has proved successful for email folder classification [3]. Considering all features binary and training Winnow using several passes through the data, Bekkerman et al. reached performance numbers comparable to standard batch learning algorithms such as SVM or Logistic Regression. In general, in order to achieve the same performance of batch algorithms, online learners have traditionally been trained using several passes through the training data.

Voting (a.k.a. Averaging) a classifier is a technique that, instead of using the best hypothesis learned so far, uses some weighted average of all hypotheses learned during the training procedure. The averaging procedure is expected to produce more stable models, which leads to less overfitting [14]. Averaging techniques have been successfully used on the Perceptron algorithm [16], but never in other online learners such as Winnow, Passive-Aggressive[10] or ROMMA[19]. In the current work, we provided a detailed performance comparison on how averaging affects the aforementioned online learners when restricted to a single learning pass. Results clearly indicate that averaging does improve the performance of single-pass online learners for non-NLP tasks.

Even though there are several available techniques to do batch feature selection for NLP tasks [13, 26], literature on Online Feature Selection is sparse. Extending such feature selection techniques to the online learning setting is not straightforward. One of the few methods that can actually add or delete features on-the-fly is Grafting [24]. Grafting is based on a a regularized risk framework, where selections and deletions are decided based on the results of an optimization procedure. Some other methods that can be adapted to perform online feature selection to some extent are described in [17]. Compared to these previous online feature selection schemes, EFS is faster and simpler to implement.

7 Conclusions

In this work we considered several methods to improve the performance of Winnow-based classifiers. We first provided a thorough performance comparison among different mistake-driven online learning algorithms, including three different versions of Winnow. One of these Winnow modifications, MBW, showed excellent performance when trained using only a single pass through the data, with numbers competitive with linear SVM for several NLP tasks.

We then considered using the averaged hypothesis technique (a.k.a. Voting) on all implemented online learners. Averaging techniques have been successfully used on the Perceptron algorithm for a while, but never in Winnow, ROMMA or other online learners. We showed that the voting technique considerably improves the classifier performance for the Winnow-based learners, and is also effective for ROMMA learner and the Passive-Aggressive learner for non-NLP datasets.

We also implemented the external regret minimization scheme proposed by Blum [4]. Using a Positive Winnow implementation, we measured measured the effect of such scheme for different datasets. In general, no consistent improvement or deterioration in performance was observed in our tests.

Finally, we proposed a new online feature selection scheme called Extremal Feature Selection (or EFS) based on the Modified Balanced Winnow learner presented. EFS is fast, simple to implement and naturally suited to the online learning setup. We showed that it can reach performance numbers comparable to traditional batch feature selection techniques such as Information Gain and Chi-Square. Results are very encouraging, indicating that EFS can be a good alternative not only to online feature selection but also to traditional batch feature selection.

References

[1] E. M. Airoldi and B. Malin, “Data mining challenges for electronic safety: The case of fraudulent intent detection in e-mails,” in *Proceedings of the Workshop on Privacy and Security Aspects of Data Mining*. IEEE Computer Society, November 2004, pp. 57–66, brighton, England.

[2] E. Airoldi, W. W. Cohen, and S. E. Fienberg, “Bayesian methods for frequent terms in text: Models of contagion and the delta square statistic,” in *Proceedings of the CSNA & INTERFACE Annual Meetings*, 2005.

[3] R. Bekkerman, A. McCallum, and G. Huang, “Categorization of email into folders: Benchmark experiments on enron and sri corpora,” CIIR, University of Massachusetts, Amherst, Tech. Rep. CIIR Technical Report IR-418, 2004.

[4] A. Blum and Y. Mansour, "From internal to external regret," in *COLT*, 2005.

[5] A. Blum, "Empirical support for WINNOW and weighted majority algorithms: results on a calendar scheduling domain," in *ICML*, Lake Tahoe, California, 1995.

[6] V. R. Carvalho and W. W. Cohen, "Learning to extract signature and reply lines from email," in *Proceedings of the Conference on Email and Anti-Spam*, Palo Alto, CA, 2004.

[7] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001, software available at http://www.csie.ntu.edu.tw/˜cjlin/libsvm.

[8] W. W. Cohen, V. R. Carvalho, and T. M. Mitchell, "Learning to classify email into "speech acts"," in *EMNLP*, Barcelona, Spain, July 2004, pp. 309–316.

[9] W. Cohen, E. Minkov, and A. Tomasic, "Learning to understand web site update requests," in *IJCAI*, Edinburgh, Scotland, 2005.

[10] K. Crammer, O. Dekel, S. Shalev-Shwartz, and Y. Singer, "Online passive-aggressive algorithms," in *NIPS*, 2003.

[11] I. Dagan, Y. Karov, and D. Roth, "Mistake-driven learning in text categorization," in *EMNLP*, Aug 1997, pp. 55–63. [Online]. Available: http://l2r.cs.uiuc.edu/ danr/Papers/Abstracts/categ.html

[12] A. Fiat and G. J. Woeginger, Eds., *Online Algorithms, The State of the Art*, ser. Lecture Notes in Computer Science, vol. 1442. Springer, 1998.

[13] G. Forman, "An extensive empirical study of feature selection metrics for text classification," *Journal of Machine Learning Research*, vol. 3, pp. 1289–1305, 2003.

[14] Y. Freund, Y. Mansour, and R. E. Schapire, "Why averaging classifiers can protect against overfitting," in *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics*, 2001.

[15] Y. Freund and R. E. Schapire, "Large margin classification using the perceptron algorithm," in *Computational Learning Theory*, 1998, pp. 209–217.

[16] ——, "Large margin classification using the perceptron algorithm," *Machine Learning*, vol. 37, no. 3, pp. 277–296, 1999.

[17] K. Glocer, D. Eads, and J. Theiler, "Online feature selection for pixel classification," in *ICML '05: Proceedings of the 22nd international conference on Machine learning*. New York, NY, USA: ACM Press, 2005, pp. 249–256.

[18] D. D. Lewis and M. Ringuette, "A comparison of two learning algorithms for text categorization," in *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, Las Vegas, US, 1994, pp. 81–93.

[19] Y. Li and P. M. Long, "The relaxed online maximum margin algorithm," in *Machine Learning*, vol. 46, 2002, pp. 361–387.

[20] N. Littlestone, “Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm,” *Machine Learning*, vol. 2, no. 4, 1988.

[21] T. Mitchell, *Machine Learning*. Mcgraw-Hill, 1997.

[22] K. Nigam, A. K. McCallum, S. Thrun, and T. Mitchell, “Text classification from labeled and unlabeled documents using EM,” *Machine Learning*, vol. 39, no. (2/3), pp. 1–32, 2000.

[23] B. Pang, L. Lee, and S. Vaithyanathan, “Thumbs up? Sentiment classification using machine learning techniques,” in *EMNLP*, 2002.

[24] S. Perkins and J. Theiler, “Online feature selection using grafting,” in *ICML*, Washington DC, 2003.

[25] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain,” in *Psychological Review*, vol. 4, 1958, pp. 386–407.

[26] Y. Yang and J. O. Pedersen, “A comparative study on feature selection in text categorization,” in *ICML*, 1997, pp. 412–420.

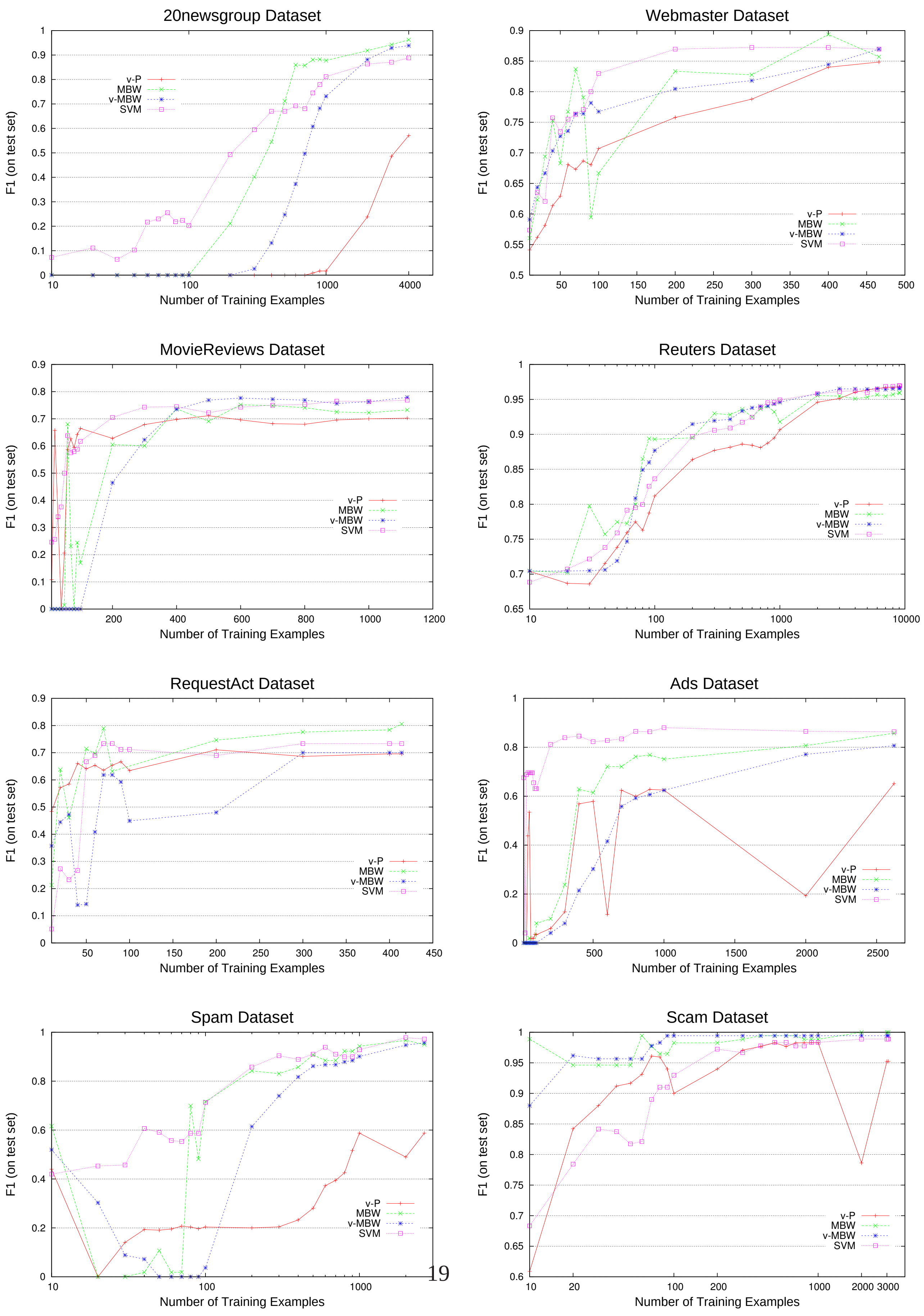


Figure 2: F1-values Learning Curves on eight NLP-like Datasets.

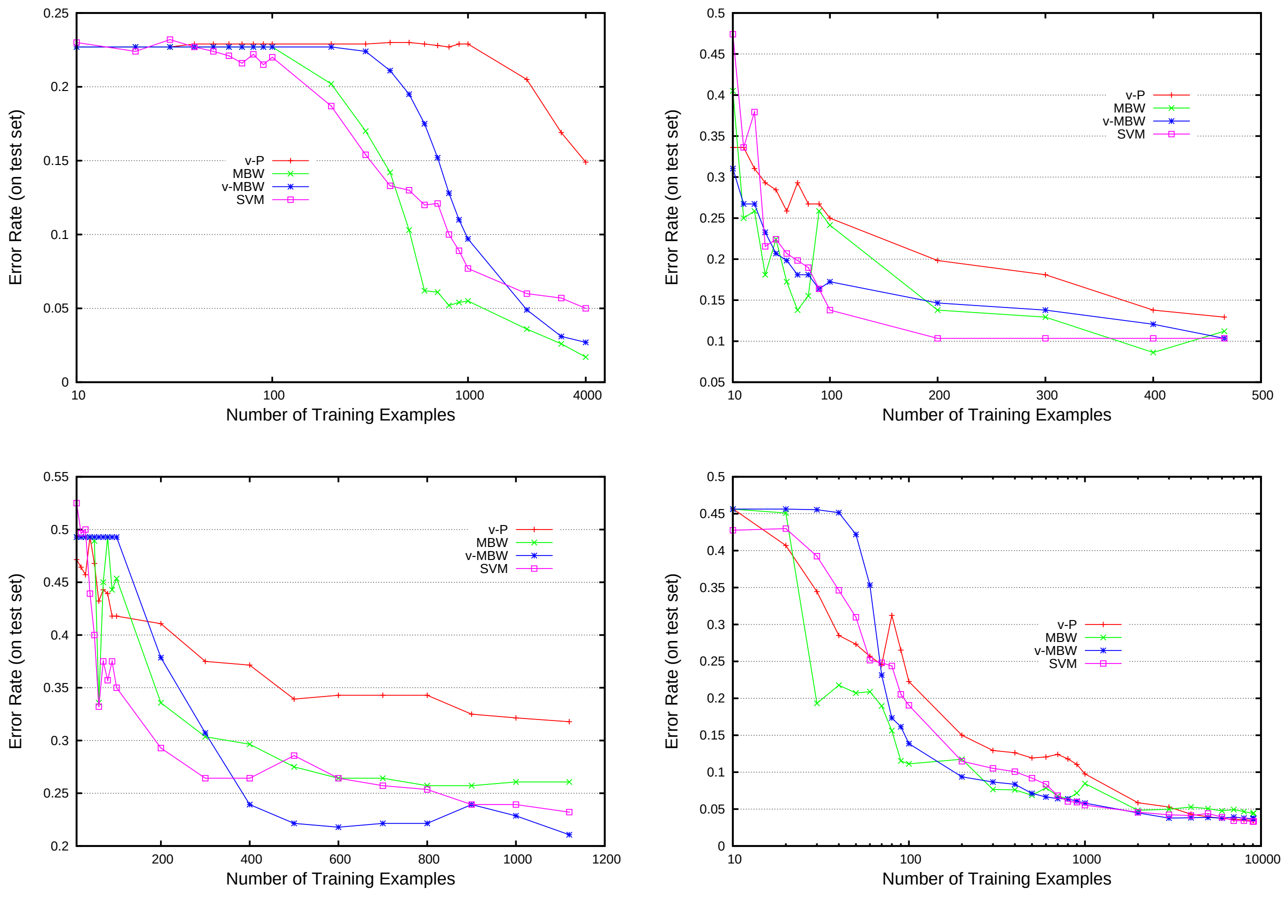


Figure 3: Test Error Rate Curves on four NLP-like Datasets.

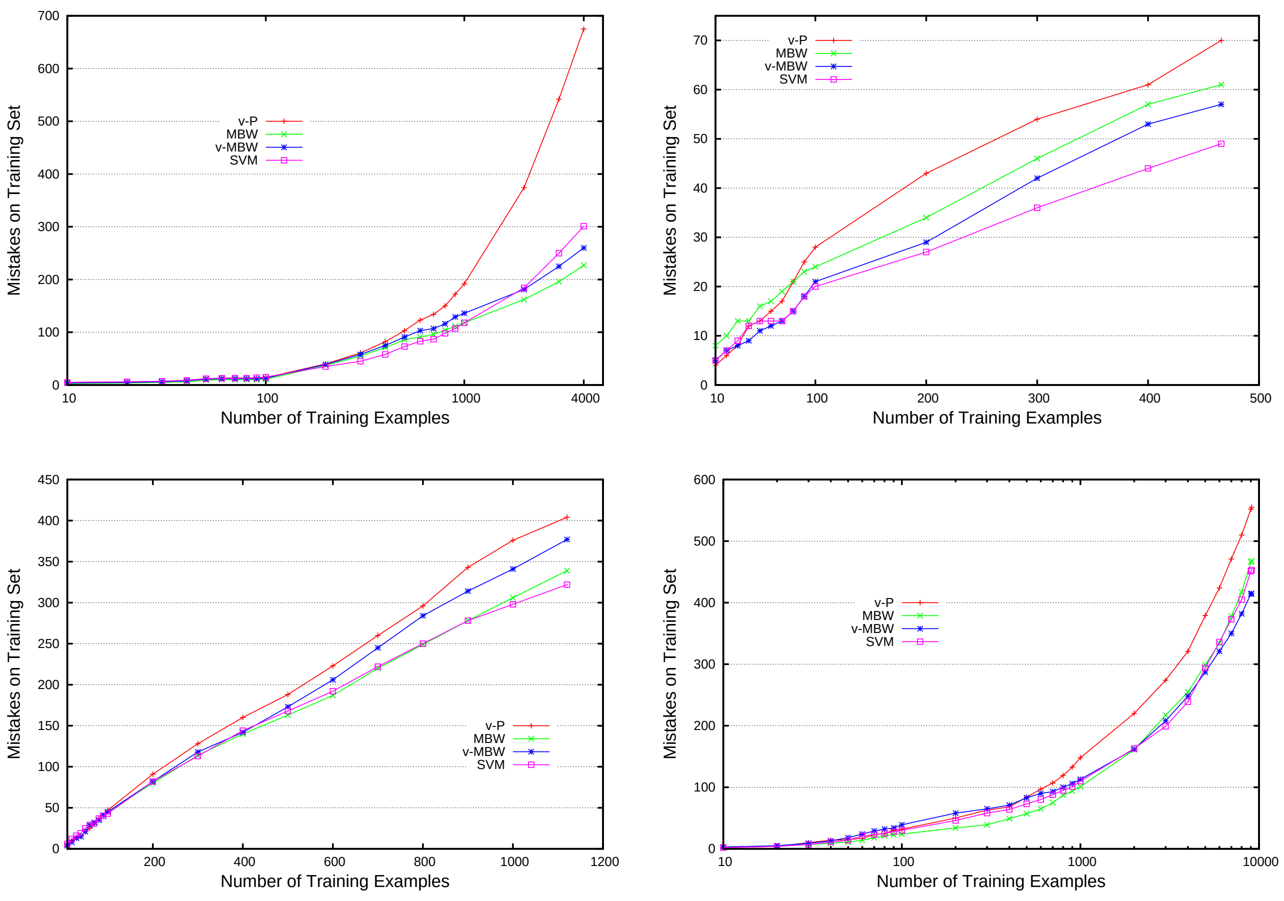


Figure 4: Training Mistakes Curves on four NLP-like Datasets.

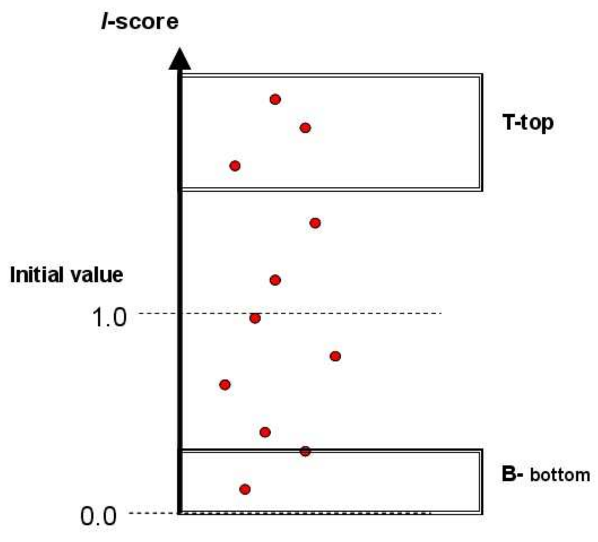


Figure 5: Selection of Meaningful Features in EFS

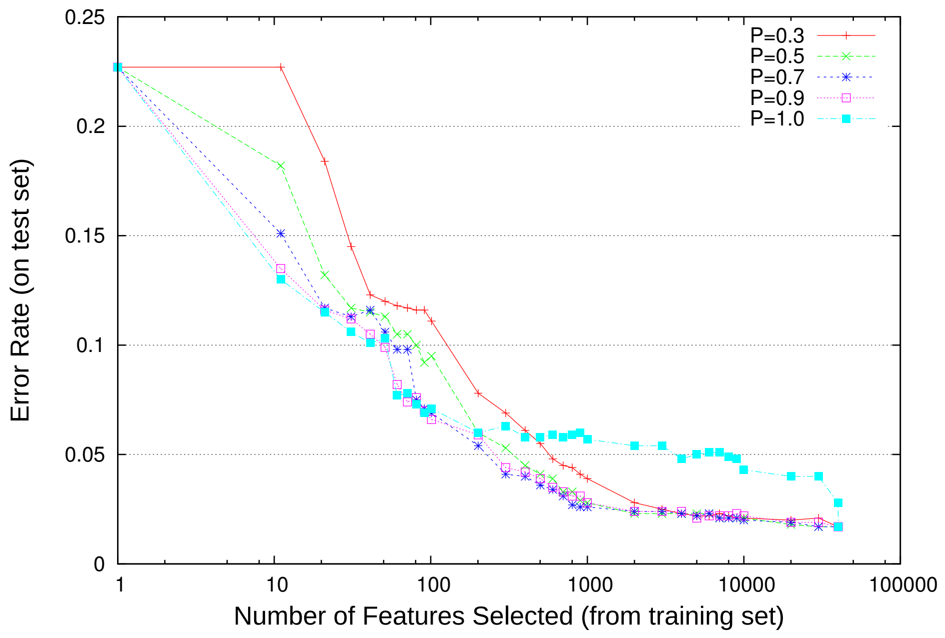
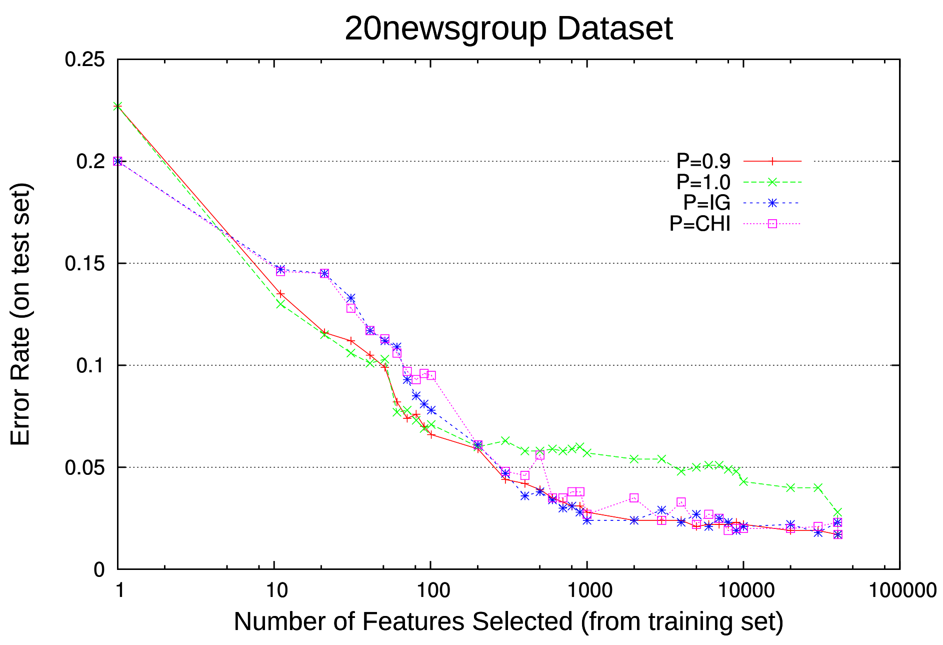


Figure 6: EFS experiment on the 20newsgroup dataset. $P = \frac{T}{T+B}$ is the fraction of selected features using high I_t^j scores.



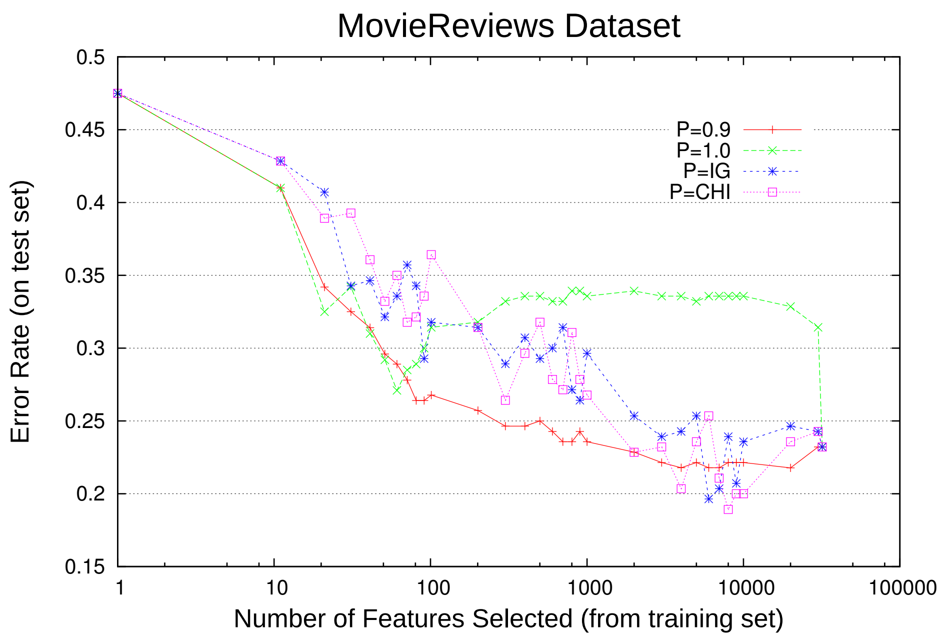
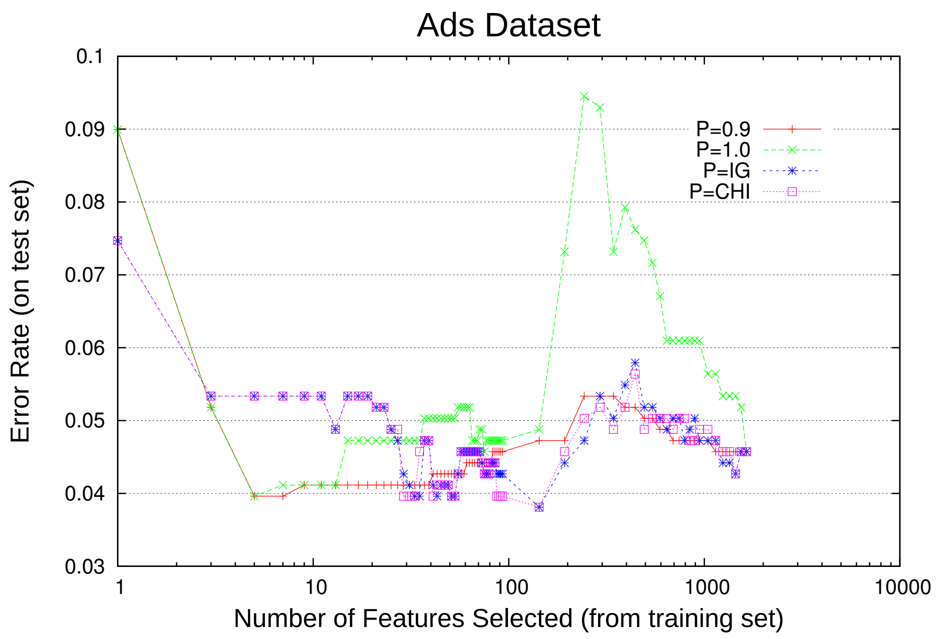


Figure 7: EFS experiments: Comparison with Information Gain and Chi-Square. $P = \frac{T}{T+B}$ is the fraction of selected features using high I_t^j scores.



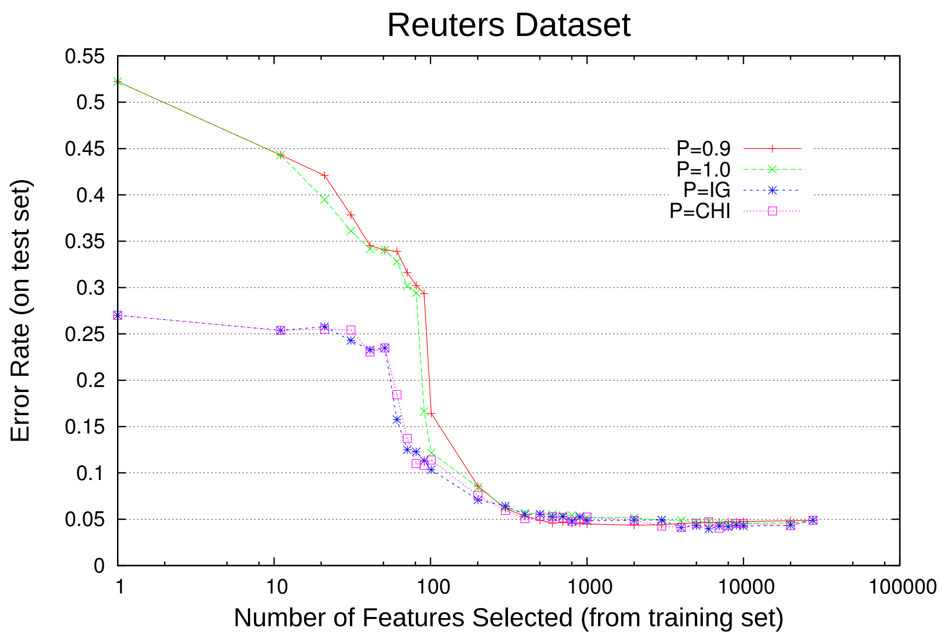


Figure 8: EFS experiments: Comparison with Information Gain and Chi-Square. $P = \frac{T}{T+B}$ is the fraction of selected features using high I_t^j scores.