Feature Selection using Multiple Streams

Paramveer S. Dhillon[1], Dean Foster[2], and Lyle Ungar[1]

[1] CIS Department, University of Pennsylvania, Philadelphia, PA 19104, U.S.A
[2] Statistics Department, University of Pennsylvania, Philadelphia, PA 19104, U.S.A

Abstract. Feature selection for supervised learning can be greatly improved by making use of the fact that features often come in classes. For example, in gene expression data, the genes which serve as features may be divided into classes based on their membership in gene families or pathways. When labeling words with senses for word sense disambiguation, features fall into classes including adjacent words, their parts of speech, and the topic and venue of the document the word is in. We present a streamwise feature selection method that allows dynamic generation and selection of features, while taking advantage of the different feature classes, and the fact that they are of different sizes and have different (but unknown) fractions of good features. Experimental results show that our approach provides significant improvement in performance and is computationally less expensive than comparable "batch" methods that do not take advantage of the feature classes and expect all features to be known in advance.

1 Introduction

For many regression or prediction tasks only a small fraction of a vast number of candidate features are predictive, and good feature selection methods can give large improvements in predictive accuracy [1]. Many methods are used to select which coefficients to include in a linear or logistic regression, generally using either an L_0 or L_1 penalty on the coefficients to force some of them to zero. Using a L_0 penalty (unlike L_1 penalization) is, in general, non-convex and so requires the use of search methods such as stepwise, stagewise, or streamwise regression. However, L_0 methods have many advantages. [2] show that L_1 never outperforms L_0 by more than a constant factor and in some cases the L_1 penalty is infinitely worse than the L_0 penalty and argue that an "approximate solution to the right problem" is generally better than "exact solution to the wrong problem". Unlike L_1 methods, L_0 penalized methods do not require the features to have a natural scale, and they lend themselves to the use of theory for selecting the magnitude of the regularization penalty, thus avoiding use of cross validation for selecting the penalty. We show below how this can be used to particular advantage when features can be naturally divided into many classes where different penalties are appropriate for the different classes.

All of the standard feature selection approaches assume that all features are in single equivalence class. This ignores the key, and useful, fact that very often

features are of different type [3]. For example, in gene expression data, the genes which serve as features may be divided into classes based on their membership in gene families or pathways. When disambiguating words, one can use adjacent words, the parts of speech of adjacent words, and the topic of the document the word is in as feature classes.

The concept of feature classes is very similar to the concept of *meta - features* which has been studied extensively in literature [4, 5]. In fact, feature classes are a special case of *meta - features* when the feature has only one meta attribute, as gene classes or topic of the word etc. in our setting. Feature classes (WSD data) compartmentalize the feature space into *blocks* as shown below in Fig 1.

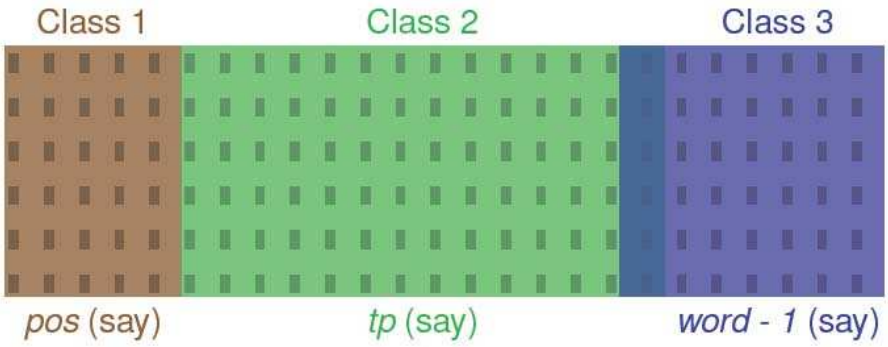


Fig. 1. *Feature Classes in data* ($tp \longmapsto$ *topic of the document*, $pos \longmapsto$ *part of speech*, $word-1 \longmapsto$ *previous word*)

More generically, starting from any set of features, one can generate new classes of features by using projections such as principle components analysis (PCA) or non-negative matrix factorization (NNMF), transformations such as log or square root, and interactions (products of features). Further "synthetic" feature classes can be created by finding clusters (e.g., using k-means) in the feature space as we do in this paper.

In all these cases, the fact that features can be grouped into classes can be used to build better feature selection methods. The basic intuition is that some feature classes will have a higher fraction of useful features than others, and that once one believes that a feature class is "good", features should be preferentially drawn from that class [3]. Since one often does not know *a priori* which feature classes are best, this needs to be learned from the data.

In this paper, we use Streamwise Feature Selection (SFS) [6], a greedy algorithm that at each iteration selects one feature to test for potential addition to the model. If the feature is found to significantly improve the model accuracy (in a way we make precise below), it is added, otherwise it is discarded and not considered again. Because streamwise feature selection allows us to dynamically decide which feature to test at each point, it is ideally suited to take advantage

of feature classes; classes which have produced more beneficial features in the past are more likely to do so in the future, and so are tapped first for candidate features. Only after they are exhausted are less productive feature classes examined.

Our contribution is to extend Streamwise Feature Selection to the case where there are multiple feature classes. When good features are unevenly distributed across the feature classes, as they generally are in practice, our new Multiple Streamwise Feature Selection (MSFS) algorithm gives superior performance over standard Streamwise Feature Selection (SFS) and widely used batch feature selection methods such as stepwise regression, lasso [L_1 penalty] [7], elastic net [$L_1 + L_2$ penalty] [8], SVM with polynomial kernel and Group Lasso [9].

The rest of the paper is organized as follows. We first introduce streamwise feature selection, on which this work is based. Details of the design and implementation of Multiple Streamwise Feature Selection (MSFS) are then described in Section 3. Finally, we present experiments showing the benefit of using multiple streams on real data, and conclude.

2 Background on Streamwise Feature Selection

Our approach is based on Streamwise Feature Selection, in which features are considered sequentially for addition to a linear model $y = \sum_i w_i x_i$. Each feature is evaluated once, and the reduction in training error resulting from adding the feature is compared against an adaptively adjusted threshold. This contrasts with batch learning methods such as SVMs, neural nets and LARS which require having all features in advance (and which do not account for feature classes). It also contrasts with stepwise regression, in which *all* features are considered at each iteration and the best feature is added. By doing sequential feature selection, we can dynamically choose which feature to consider at each point based on what has worked so far, and we can automatically adjust the threshold for feature inclusion. This allows us, among other benefits, to have streams of features that are dynamically generated, such as interactions between those features which have been already added to the model. (Each time a feature is selected, products of it and other features are added to the candidate feature stream.)

A variety of penalties have been used on weights in a linear or logistic regression setting to avoid overfitting. Ridge regression (a L_2 penalty on the weights w), has the advantage of having a closed form solution, but does not do feature selection and thus performs poorly when there are vastly more spurious than true features. Lasso regression (a L_1 penalty on the weights) is convex, and permits a relatively efficient batch optimization algorithm to be used, but requires cross validation to determine the best penalty, and does not scale to the size of feature sets we are interested in addressing. Lasso is computationally very expensive, so the authors [7] prescreen the features to keep the top 1,000 features (based on p - value) and run Lasso on them, which will, in certain cases, miss many highly predictive features. Lasso also does not support dynamically generating and testing new features. We thus use L_0 penalized regression, which

requires search (stepwise or streamwise) over the features to be added. Although finding an optimal solution is in theory NP hard, good solutions are almost always found by very simple search strategies, and the penalty for adding features can be chosen by theory. Among Stepwise and Streamwise regression, we chose Streamwise feature selection (SFS) as it has many attractive properties which we elucidate throughout the paper.

A variety of different penalties λ have been used when minimizing $\sum(y_i - wx_i)^2/2\sigma^2 + \lambda |w|_0$. (Note that $|w|_0$ is just the count of how many features are included in the model.) The AIC penalty uses $\lambda = 2$, BIC uses $\lambda = log(n)$ (where n is the total number of observations) and is optimal if there are far more observations than features, and RIC (also called Bonferroni) [10] uses a penalty of $2log(p)$ (where p is the total number of features) and is optimal if only very few features are expected to be added (otherwise it is generally overly conservative). All of these fixed-penalty methods fail (either due to under or overfitting) in the limit of infinite numbers of features considered, and none lend themselves to taking advantage of multiple feature classes.

SFS either using alpha-investing as presented here, or in its information theoretic form [6], dynamically adjusts the penalty for adding a new feature by changing the threshold on the error reduction required for adding a new feature.

The alpha-investing algorithm is given in Algorithm 1. The threshold, α, corresponds to the probability of including a spurious feature (one which increases error on a hypothetical test set) at step i. It is adjusted using the wealth, w, which represents the current acceptable number of future false positives. Wealth is increased when a feature is added to the model (presumably correctly, and hence permitting more future false positives without increasing the overall false discovery rate). Wealth is decreased when a feature is not added to the model, in order to save enough wealth to add future features.

More precisely, a feature is added to the model if its p-value is less than α. The p-value is the probability that a feature coefficient would be judged to be non-zero when it is in fact zero. It is computed as $e^{(-(Error_{\{model\}\cup\{x\}} - Error_{model})/2\sigma^2)}$, where $Error_{model}$ is the root mean squared error on the training set using the features in the set model, $Error_{\{model\}\cup\{x\}}$ is the error after adding the feature x, and the variance σ^2 is estimated as $Error_{model}/n$.

The idea of α-investing is to adaptively control the threshold for adding features so that when new (probably predictive) features are added to the model, one "invests" α increasing the wealth, raising the threshold, and allowing a slightly higher future chance of incorrect inclusion of features. We increase wealth by $\alpha_\Delta - \alpha$. Note that when α is very small, this increase amount is roughly equivalent to α_δ. Each time a feature is tested and found not to be significant, wealth is "spent", reducing the threshold so as to keep the guarantee of not adding more than a target fraction of spurious features. There are two user-adjustable parameters, α_Δ and w_0, which can be selected to control the false discovery rate; we always set both of them to 0.5.

Streamwise Feature Selection, and the novel extension to multiple streams presented below, provides guaranteed bounds on the ratio of $E(N)$, the expected

Algorithm 1 SFS using Alpha-investing

```
1: w = w_0; // initial probability of false positives.
2: model = {}; // initially no features in model
3: i = 1; // index of features
4: while features remain do
5:    x = get_new_feature(); // generate next feature
6:    α = w/2i;
7:    // is p-value of new feature below threshold?
8:    if (get_p_value(x, model) ≤ α) then
9:       // accept
10:      add_feature(x, model); // add x to the model
11:      w := w + α_Δ − α;
12:   else
13:      // otherwise, reject
14:      w := w − α; // reduce wealth
15:   end if
16:   i := i + 1;
17: end while
```

number of spurious features included in the model to $E(M)$ the expected number of beneficial features included in the model. This bound on $E(N)/E(M)$ is very similar to bounding the false discovery rate. FDR), which is $E(N/M)$ [11].

Theorem 1: *Let M_i be the number of correct (beneficial) features included in the model, and let N_i be the number of spurious features included in the model and let w_i be the wealth, at iteration i, also let α_Δ be a user selected parameter. Then:*

$E(N) < (\alpha_\Delta E(M) + w_0)/(1 - \alpha_\Delta)$

Proof:

The proof relies on the fact that

$$S_i \equiv (1 - \alpha_\Delta)N_i - \alpha_\Delta M_i + w_i$$

is a super-martingale [12] i.e. S_i is, in expectation, non increasing in each iteration: $E(S_i) \leq S_{i-1}$. Thus

$$E(S_i) \leq S_0$$

but since we start out with $N_i = 0$ and $M_i = 0$

$$E((1 - \alpha_\Delta)N_i - \alpha_\Delta M_i + w_i) \leq w_0$$

Also, since $w_i > 0$ (by assumption), therefore

$$E((1 - \alpha_\Delta)N_i - \alpha_\Delta M_i) < w_0$$

The proof that S_i is a super-martingale follows from the cases when the feature is or not added to the true and estimated model. □

The result can we re-written as:

$$E(N_i)/(E(M_i) + w_0/\alpha_\Delta) < \alpha_\Delta/(1 - \alpha_\Delta)$$

For the case that we use in all our experiments, $\alpha_\Delta = w_0 = 1/2 << E(M)$, this reduces to $E(N) < E(M)$, or simply that we promise to add more good features than bad ones. Note that "spurious" and "beneficial" indicate whether adding a features to the current model will decrease or increase accuracy on a (hypothetical) test set.

3 Multiple Streamwise Feature Selection (MSFS)

Streamwise feature selection has a natural extension to multiple streams [3]. Each set of features (feature class) is taken to be its own stream, with its own index, keeping track of how many features from that stream have been tested, and its own wealth, measuring how successful the stream has been in producing useful features. Thus, a stream gains or loses wealth based only on how successful it has been in producing beneficial features. The question of which feature to select next for testing is easily resolved: at each iteration, the next feature is taken from the stream with the most permissive threshold, i.e., from the feature class with the highest probability of producing a beneficial feature, i.e. the feature class having maximum wealth.

The MSFS algorithm is given in Algorithm 2. It is quite similar to the simple SFS algorithm presented above, but keeps track of wealth and features tested separately for each stream, j. In order to continue to guarantee against overfitting, each of the k different feature streams is only given w_0/k initial wealth. The function $get\_new\_feature()$ can just get the next feature in the set (the $i_j th$ feature in stream j) or, as discussed below, it can dynamically generate new features. As always, we set the constants α_δ and w_0 to 0.5.

If a fraction $1/(m_j - 1)$ of the features are added to stream j, each additional feature changes the wealth w_j by $\alpha_\Delta/m_j - w/2i$, and w_j will approach α_Δ/m_j. When good features are concentrated in one stream, then that stream will soon have the highest wealth, and features will be drawn preferentially from that stream. If there are k (equally sized) streams with all good features in a single stream, then one will only need to consider slightly more than a fraction $1/k$ of the features. (Very few features in the other streams need to be considered.) The wealth in the good stream approaches $\alpha_\Delta/(p/kq)$ (where q is the number of true features and p is the total number of features), rather than the value of $\alpha_\Delta/(p/q)$ in the single stream setting, thus allowing more features with marginal statistical significance to be retrieved from the good stream. Similar, but less strong benefits occur in the more realistic case where the streams simply contain different densities of good features.

[3] The terms "streams" and "feature classes" are used interchangeably throughout the paper and they have the same meaning in this paper; the only difference being that "stream" is a term that is specific to streamwise regression whereas "feature class" is a general term that refers to compartmentalization of the feature matrix (data).

Algorithm 2 MSFS using Alpha-investing

```
for j = 1 to k do
    w_j = w_0/k; // initial wealth for j-th stream
    i_j = 1; // index of features for j-th stream
end for
model = {}; // initially no features in model
while features remain do
    // select next stream
    j = argmax_j(w_j/i_j); // over all streams with remaining features
    x = get_new_feature(j, i_j); // generate new feature on stream j
    α = w_j/2i_j;
    // is p-value of new feature below threshold?
    if (get_p_value(x, model) ≤ α) then
        // accept
        add_feature(x, model); // add x to the model
        w_j := w_j + α_Δ − α; // increase wealth
    else
        // otherwise, reject
        w_j := w_j − α; // decrease wealth
    end if
    i_j := i_j + 1;
end while
```

Since we can order the features within a stream, streams such as PCA components (arranged from largest to smallest eigenvalue) will quickly be recognized. The stream with the original features will learn a threshold that is more stringent than the PCA stream, but much more permissive than the $p^2/2$ interaction terms, which is likely to be explored last.

In the worst case, when the features are randomly dispersed across the streams, features are drawn sequentially from each of the feature classes (streams), all of which have success at the same rate, and the resulting penalty (remembering that each stream starts out with w_0/k of the wealth) looks asymptotically exactly like running a single stream. Having $1/k$ of the wealth in each stream will reduce the chance of finding features very early in the streams (i.e., among the first few features tried), but the effect of the initial allocation of wealth is rapidly dissipated, and the wealth approaches an asymptote determined by α_Δ and by the fraction of features found to be significant.

It is not easy to get comparable benefits from grouping features into classes using batch feature selection algorithms, whether they be stepwise regression or LARS/elastic net. Since all features are considered simultaneously in batch algorithms, one has to "pay the penalty" for looking at all of them when avoiding overfitting. This difficulty can be overcome by applying a different weight to each of the groups of variables, but each of these weights would then have to be chosen via cross validation–basically requiring searching for a vector of penalties in a space of $\Re^k$.

3.1 Interleaving Feature Generation and Selection

The "streamwise" view supports flexible ordering on the generation and testing of features. Features can be generated dynamically based on which features have already been added to the model [4]. Note that the theory provided above is independent of the feature generation scheme used. All that is required is a method of generating features that does not look at the y values, and an estimation package which given a proposed feature for addition to the model returns a p-value for the corresponding coefficient or, more generally, the change in likelihood of the model resulting from adding the feature. One can also test the same feature more than once, as we do in this paper (10 passes of Streamwise Feature Selection).

New features can be generated in many ways. Each way produces a new feature class for use in MSFS. For example, in addition to the p original features, p^2 pairwise interaction terms can be formed by multiplying all p^2 pairs of features together. In practice, we generate three interaction streams: (1) interactions of features that already been selected with themselves (2) interactions of the selected features with the original features, and (3) all interactions of the original features. This requires dynamic generation of the feature stream, since the interaction terms (1) and (2) can not be specified in advance, as they depend on which features have already been selected. As shown in Section 4, the dynamic feature generation and selection schemes, namely (1) and (2) above learn significantly more accurate models on real data sets compared to the approaches which do not use these dynamic interactions.

Interaction terms are one example of a more general class of *generated* features, including features formed from transformations of the original features (square root, log, etc.), or combinations of them including, for instance, PCA. Such generated features frequently lead to substantially better predictive models, but it is not obvious which of the transformations will be most useful. By putting each into its own stream, and using MSFS, one can try many transformations are relatively little cost. In contrast, in a conventional batch method, one would need to look at all the features in all the streams, at significant computational cost and, worse, at the cost of statistical power of needing to use a larger penalty to control against overfitting. Including separate feature classes for algebraic transformations and PCAs of original features, gives improvement in predictive power as we elucidate in Section 4.

Another motivation for dynamic generation and testing of features arises in Statistical Relational Learning (SRL) tasks where tens or hundreds of thousands of potentially predictive features are generated by crawling through a relational database [13]. In this case, since the potential set of features can be really large, it makes sense to interleave the generation of features with the assessment of model improvement i.e. feature selection. This way we are pruning the search over the

[4] One cannot use the coefficients of the features that were not added to the model, since streamwise regression does not include the cost of coding these coefficients, and so this *would* lead to overfitting. One can, of course, use the rejected features themselves in interaction terms, just not their coefficients.

feature space to only those features which are likely to lead to improvement, and thus make a potentially intractable search over a billion features, tractable.

4 Experimental Results

In this section, we evaluate the MSFS algorithm on real data to gauge its effectiveness. We ran it on a set of ten word sense disambiguation (WSD) verb datasets [14] and two NIPS datasets [1]. The WSD datasets already had feature classes, as described above. In contrast, the NIPS datasets did not come with feature classes identified. So, we created synthetic feature classes by grouping (using k-means clustering) the features, by taking the top 50 PCAs, and by computing the square of each of the features in the original set. In all the experiments we compare MSFS with standard streamwise feature selection (SFS), stepwise feature selection with an RIC penalty (also called a "Bonferroni correction"), Lasso, Elastic Nets (EN), SVM with cross validated polynomial kernel [5] and with Group Lasso/ Multiple Kernel Learning (GL/MKL), which minimizes a mixed L_1/L_2 norm. GL/MKL makes use of feature classes but induces sparsity only at the level of feature classes and not at the level of individual features.

Besides the accuracies, we also compare the running times of the above algorithms. All the experiments were run on a 3 GHz computer with 8 GB of RAM.

4.1 Evaluation on Word Sense Disambiguation (WSD) Datasets

We chose a set of 10 ambiguous verbs with a rich set of contextual features [14] for our first evaluation. The size of the WSD data and other relevant information are summarized in Table 1. The data consists of hundreds of observations of Noun-Noun collocation, Noun-adjective-preposition-verb (syntactic relations in a sentence) and Noun-Noun combinations (in a sentence or document). Typical feature classes include: tp (topic of the document), pos (part of speech of the verb), word-1(previous word), sub (the subject of the verb), dobj (the direct object of the verb), dobjsyn (dobj's wordnet synsets) and morph (verb morphology) etc.

One stream was generated for each different feature class, so the number of streams is the same as the number of feature classes. Some streams have exactly one feature, while others have thousands of features. For MSFS this is a natural setting but SFS does not use feature classes so, for SFS we simply collapse all the multiple streams into a single stream, with streams having the fewest features placed at the front. This ordering helps significantly since streams with small number of predictive features should be presented to the streamwise selection algorithm first to avoid being overlooked as the wealth (α-value) decreases over time. Then, in the end we have a stream (feature class) for dynamic interactions

[5] Standard SVMs do not do feature selection but are just included for completeness of results.

of the selected features with themselves and with all other features, however this stream (feature class) only gets created dynamically as and when new features are added into the model. We randomly permute the features within each feature class so that no method gets an unfair advantage by exploiting the ordering of features. (In practice, SFS or MSFS can sometimes gain significant advantage by, for example ordering the features by their variance or by the fraction of times they are nonzero.)

For Group Lasso/Multiple Kernel Learning, we used a set of 13 candidate kernels, consisting of 10 Gaussian Kernels (with bandwidths $\sigma = 0.5 - 20$) and 3 polynomial kernels (with degree 1-3) for each feature class as is done by [15]. In the end the kernels which have non zero weights are the ones that correspond to the selected feature classes. Since GL/MKL minimizes a mixed L_1/L_2 norm so, it zeros out some feature classes. (Recall that GL/MKL gives no sparsity at the level of features within a feature class). The Group Lasso [9] and Multiple Kernel Learning are equivalent, as has been mentioned in [16], therefore we used the *SimpleMKL* toolbox [15] implementation for our experiments.

For Lasso (L_1 penalty) and Elastic Nets ($L_1 + L_2$ penalty), we used their standard LARS implementation [7] (which performs an internal cross validation to select the best regularization penalty (λ)) and screened the features to keep the "best" 800, as it is not possible to run these methods on big datasets. [7] also do a similar screening for the same reason.

For SVM we used its standard LIBSVM implementation [17] and performed cross validation to select the best value of "gamma" parameter [17].

Table 1. Word Sense Disambiguation Datasets.

Verb	Observations	Features	Feature Classes
acquire	101	1081	43
care	131	621	40
climb	84	676	41
fire	132	1217	43
add-1	320	2583	42
expand	222	2144	42
allow	344	2657	41
drive	191	1584	43
identify	102	964	43
promise	111	929	41

The accuracies of various methods, presented as 10-Fold CV [6] classification accuracies are shown in the Table 2 and the running times are compared in Fig. 2.

As is obvious from Table 2, MSFS (I), i.e. MSFS with feature set augmented with dynamically generated interactions of selected features with themselves and of selected features with all other features i.e. interaction streams (1) and (2) as mentioned in Section 3.1, gives the best predictive accuracy in 7 out of 10 cases, and in the remaining 3 cases it is the second best method. (All the results are statistically significant at 5% significance level in paired t-test.). This performance improvement can be explained by the fact that the augmented features (i.e., the dynamic interaction terms) contain highly predictive features, which when added to the model give improved predictive power. It is important to note that we can not generate such "dynamic" interactions for any of the other methods as these interaction terms are not known beforehand and are generated dynamically. The only way we can generate these interaction terms for the other methods is by considering all the $p^2/2$ "static" interaction terms (where p is the total number of features); which becomes computationally infeasible even if we have $p = 2000$ as in that case we will have 2 million interaction terms. The ability to generate interaction terms on-the-fly is at the heart of MSFS (I) and is its biggest advantage over the other "batch" methods.

The benefits of including interaction terms can be corroborated by the fact that in most of the cases MSFS (I) selects more features on average than MSFS, so these "extra" features come from the dynamic interaction terms as otherwise the two methods are the same. In two cases MSFS (I) puts in fewer features than MSFS, but still it gives better predictive performance due to the fact that the increased size of the feature set permits spending of less wealth on spurious features (false positives), unlike MSFS and SFS which tend to put in more features. SFS generally performs worse than MSFS due to the fact that it is not taking advantage of the presence of feature classes in that data and thus distributes the entire wealth evenly over the complete feature stream.

It is worth noting that other feature selection methods such as Stepwise RIC (L_0 penalty), Lasso (L_1 penalty) and Elastic Nets ($L_1 + L_2$ penalty) put in too many features, as is evident from the numbers and the fact that in only one out of ten cases ("allow") does one of these "batch" methods beat MSFS. It turns out that in that case MSFS (I) puts in too many spurious features. In two other cases ("expand", "identify") these methods manage to have the same accuracy as MSFS. Multiple Kernel Learning (MKL), does well on all the verbs, just being a marginally worse than the best method. Its good performance can be attributed to the fact that it is a large margin method and explicitly uses optimization to find the feature classes that should be included in the model. However, like the other "batch" methods, it suffers the drawback that it cannot handle dynamic generation of interaction terms. Also, since GL/MKL penalize a

[6] These cross validation accuracies are "proper" test acuracies and no parameters have been tuned on them. There were no standard train-test splits for our data so instead of arbitrarily making such splits, we report 10-Fold CV accuracies.

mixed (L_1/L_2) norm, so they induce sparsity only at the level of feature classes and tend to put all the features in the selected feature class in the model.

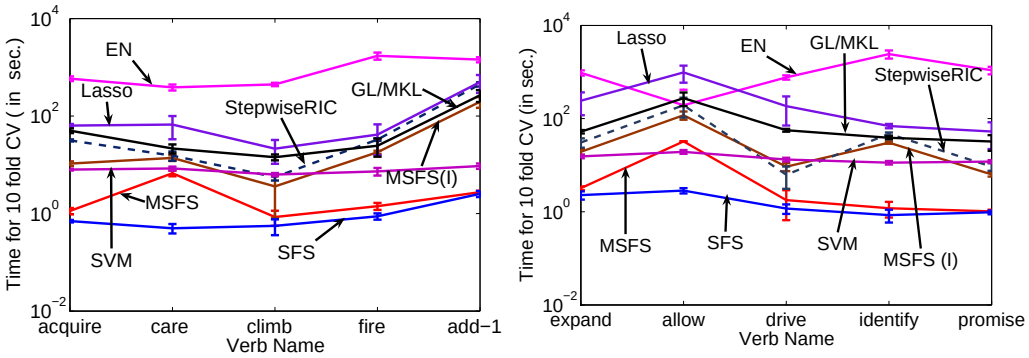
Next we look at the running times and computational complexities for the various methods as shown in Fig. 2. As is obvious from the plots, Lasso, Elastic Nets and GL/MKL are the most computationally expensive methods. This can be explained by the fact that all these methods explicitly solve optimization problems which are very expensive, namely finding the regularization paths for Lasso and Elastic Nets [7] and solving a Quadratic Program and prior computation of all the kernels [15] for MKL.

On the other hand, MSFS (I), MSFS and SFS are extremely cheap as they just make a fixed number of passes (multi pass SFS) through the feature set [6]. In general, their order of complexity is $O(kN)$ where N is the number of features and k is the number of passes made (10 in our case). The time complexity for Stepwise RIC is $O(N^2)$ and other methods i.e. the ones involving optimization have worst complexities i.e. $\geq O(N^2)$.

Among MSFS (I), MSFS and SFS, MSFS(I) takes more time as it needs to generate and test the dynamic interaction terms for inclusion in the model.

Table 2. 10 Fold CV accuracies of various methods on the WSD dataset (10 verbs). *Note:* **The cross validation accuracies are "proper" test acuracies and no parameters have been tuned on them.** *Note*: MSFS (I) means MSFS with dynamic interaction terms and ((#*f*) means the avg. number of features selected per fold). Standard SVMs do not do feature selection.

Method	acquire	care	climb	fire	add-1
	$\mu \pm \sigma$ (#*f*)	$\mu \pm \sigma$ (#*f*)	$\mu \pm \sigma$ (#*f*)	$\mu \pm \sigma$ (#*f*)	$\mu \pm \sigma$ (#*f*)
MSFS(I)	**97.0±0.9 (3.4)**	**96.9±0.4 (5.5)**	**97.5±0.5 (0.5)**	**98.5±1.1 (1.6)**	95.9±0.6 (13.3)
MSFS	91.1±1.9 (2.3)	95.5±0.7 (1.3)	92.5±1.3 (1.3)	96.3±1.0 (8.1)	**96.6±0.9 (5.3)**
SFS	93.1±0.2 (3.5)	95.6±0.1 (2.5)	95.0±1.0 (3.5)	96.3±2.0 (4)	95.0±0.3 (16.9)
Stepwise RIC	93.1±0.3 (5.1)	93.1±1.3 (12.3)	88.8±3.6 (3.7)	95.5±1.4 (3)	91.9±2.3 (17.2)
Elastic Nets	90.5±4.1 (6.1)	96.1±0.2 (24.1)	92.5±1.1 (91)	95.4±2.2 (107.8)	93.4±0.5 (1)
Lasso	90.0±2.9 (15.2)	85.4±1.9 (35.8)	88.8±2.1 (84.9)	93.1±3.1 (106.7)	93.8±0.2 (1)
GL/MKL	96.0±0.1 (50.3)	96.0±0.3 (21.7)	95.9±0.6 (11)	97.5±0.3 (12)	91.3±2.5 (1952)
Poly. SVM	94.1±0.6 (-)	95.2±0.4 (-)	93.6±0.7 (-)	96.0±0.5 (-)	92.1±1.1 (-)
Method	expand	allow	drive	identify	promise
	$\mu \pm \sigma$ (#*f*)	$\mu \pm \sigma$ (#*f*)	$\mu \pm \sigma$ (#*f*)	$\mu \pm \sigma$ (#*f*)	$\mu \pm \sigma$ (#*f*)
MSFS (I)	**99.5±0.2 (3.4)**	92.8±2.0 (12.1)	**98.4±0.8 (5.7)**	**99.0±0.4 (7.2)**	94.5±1.7 (2.1)
MSFS	97.8±0.6 (12)	92.7±1.8 (3.9)	97.4±1.1 (8.5)	98.0±0.3 (3.1)	**95.5±1.4 (1.7)**
SFS	98.7±0.1 (9.8)	91.8±1.0 (9.5)	95.8±1.7 (7.8)	97.0±0.6 (7.2)	92.8±2.0 (3.1)
Stepwise RIC	96.4±0.4 (4.3)	88.5±3.1 (22.4)	92.1±3.1 (6.0)	**99.0±0.5 (1.9)**	88.2±3.1 (6.4)
Elastic Nets	99.1±0.3 (106.9)	**93.9±0.9 (5.8)**	96.9±1.0 (1.3)	89.0±3.1 (41.2)	91.9±2.7 (4.8)
Lasso	**99.5±0.3 (81.9)**	89.1±2.4 (69.9)	92.1±2.4 (18.1)	86.0±2.9 (10)	88.3±4.1 (20.6)
GL/MKL	97.7±0.7 (53)	92.6±2.3 (2294)	97.5±0.4 (28)	97.3±0.6 (1)	90.4±3.2 (232)
Poly. SVM	98.1±0.3 (-)	91.6±0.1 (-)	94.4±0.5 (-)	96.5±0.8 (-)	92.6±0.9 (-)



(a) (b)

Fig. 2. Graphs showing total running times for 10 fold CV (in seconds) for various methods for WSD Datasets. *Note:* The y-axes are drawn on log scale

4.2 Evaluation on NIPS 2003 Datasets

Next we test our approach on two of the NIPS 2003 [1] datasets namely *arcene* (100 observations, 10000 features) and *gisette* (100 observations, 5000 features). We selected these datasets to demonstrate that our approach works well in the case in which there are no naturally occuring feature classes in data. We want to underscore the fact that augmenting the feature set with PCAs and algebraic transformation like "squares" of the features can give improved performance. We did not run experiments on the remaining 3 NIPS datasets (i.e. *dexter, madelon, dorothea*) as they are either synthetic or have sparse or binary features, hence certain transformations like "squares" transformation won't be meaningful.

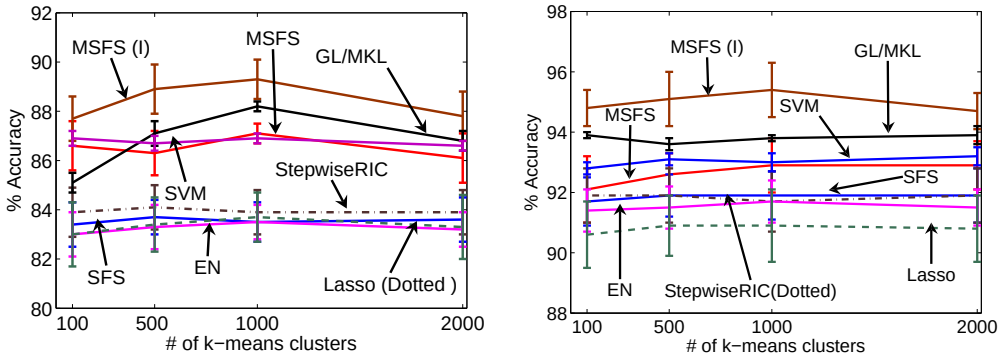
We created synthetic feature classes for these datasets by clustering the features using k-means with Euclidean distance. It turns out that features in each class are similar, but sufficiently different to provide non - redundant signal for prediction of the responses (Y's). In addition to this we also added feature classes corresponding to the top 50 PCAs and the "squares" of the features. This gave $k + 2$ feature classes in all, where k is the number of k-means clusters.

The results are shown in Table 3 for the case when $k = 1000$ clusters. Fig. 3 shows that the accuracy is not a strong function of the number of clusters, however reasonable clusters must exist, for e.g. for the NIPS "dexter" dataset most of the points fell in a single cluster, and clustering did not help. This skewed clustering for "dexter" can be explained by the fact that it is *sparse* hence we did not use it and instead used "arcene" and "gisette" both of which are *dense*.

Due to paucity of space we have not shown the running times of various methods for the NIPS datasets, but the trend is similar to the one for WSD data as the underlying optimization criterion and hence the computational complexities of the various methods remain the same.

In both the NIPS datasets, MSFS (I) and MSFS select features from the feature class corresponding to top 50 PCAs, but do not select any features from "squares" feature class, which implies that PCA is a good transformation to

use as it gives highly predictive features whereas "squares" is not. Generating and testing these extra PCA and "squares" feature classes incur only a minor overhead as the number of total features is only roughly doubled. For both the datasets, MSFS(I) shows the highest accuracy for all cluster sizes followed by Group Lasso/ MKL as is obvious from Fig. 3 (All the results are statistically significant at 5% significance level in paired t-test).



(a) (b)

Fig. 3. Accuracy as a function of # clusters for (a) *arcene* and (b)*gisette* datasets

Table 3. Results on the NIPS datasets (10 Fold CV Accuracies).*Note:* **The cross validation accuracies are "proper" test acuracies and no parameters have been tuned on them**. *Note:* MSFS (I) means MSFS with dynamic interaction terms and ($\#f$) means the avg. number of features selected per fold). Cluster # with best accuracies shown here. Standard SVMs do not do feature selection.

Method	arcene (1000) $\mu \pm \sigma$ ($\#f$)	gisette (1000) $\mu \pm \sigma$ ($\#f$)
MSFS (I)	**89.3±0.8 (5.3)**	**95.4±0.9 (23.33)**
MSFS	87.1±0.4 (4.4)	92.9±0.9 (20)
SFS	83.5±0.8 (4.4)	91.9±0.8 (18)
Stepwise RIC	83.9±0.9 (4.2)	91.7±1.0 (6.33)
Elastic Nets	83.5±0.7 (20.1)	91.7±0.7 (92.67)
Lasso	83.7±1.0 (13.4)	90.9±1.2 (92.33)
Group Lasso/MKL	88.2±0.2 (225)	93.8±0.1 (48.97)
SVM (Poly. Kernel)	86.9±0.2 (-)	93.0±0.3 (-)

5 Conclusion

We have shown both in theory and in practice that exploiting the fact that features come in classes can lead to significant gains in predictive accuracy. The Multiple Streamwise Feature Selection (MSFS) algorithm is simple, requiring less than a page of Matlab code, and extremely fast, since each potential feature is considered only once. Moreover MSFS can be extended to include dynamically generated interaction terms. Doing so generally gives significant improvement in performance accuracy over batch methods which would need to include all $p^2/2$ interaction terms. MSFS would be even more helpful in tasks such as Statistical Relational Learning (SRL), where thousands of feature classes containing billions of new features can be generated by doing sequentially more complex queries against a database, with each query generating a new feature class. Also, as demonstrated above, MSFS allows one to test whether new transformations/projections of the features will be of any help or not by incurring only a little overhead on the overall computational cost and, even more importantly, little loss of statistical power to avoid overfitting. Finally, MSFS is computationally much less expensive than the state of the art "batch" methods.

References

1. Guyon, I. In: NIPS 2003 Workshop on Feature Extraction and Feature Selection. (2003)
2. Lin, D., Pitler, E., Foster, D.P., Ungar, L.H.: In defense of ℓ_0. In: Sparse Optimization and Variable Selection Workshop ICML/COLT/UAI. (2008)
3. Dhillon, P.S., Foster, D., Ungar, L.: Efficient feature selection in the presence of multiple feature classes. In: International Conference on Data Mining (ICDM). (2008) 779–785
4. Eyal Krupka, Amir Navot, N.T.: Learning to select features using their properties. Journal of Machine Learning Research **9** (2008) 2349–2376
5. Lee, S.I., Chatalbashev, V., Vickrey, D., Koller, D.: Learning a meta-level prior for feature relevance from multiple related tasks. In: ICML '07, New York, NY, USA, ACM (2007) 489–496
6. Zhou, J., Foster, D.P., Stine, R.A., Ungar, L.H.: Streamwise feature selection. Journal of Machine Learning Research **7** (September 2006) 1861–1885
7. Efron, B., Hastie, T., Johnstone, L., Tibshirani, R.: Least angle regression. Annals of Statistics **32** (2004) 407–499
8. Zou, H., Hastie, T.: Regularization and variable selection via the elastic net. Journal Of The Royal Statistical Society Series B **67**(2) (2005) 301–320
9. Yuan, M., Lin, Y.: Model selection and estimation in regression with grouped variables. Journal of the Royal Statistical Society: Series B (Statistical Methodology) **68**(1) (February 2006) 49–67
10. Foster, D.P., George, E.I.: The risk inflation criterion for multiple regression. The Annals of Statistics **22**(4) (1994) 1947–1975
11. Benjamini, Y., Hochberg, Y.: Controlling the false discovery rate: A practical and powerful approach to multiple testing. Journal of the Royal Statistical Society. Series B (Methodological) **57**(1) (1995) 289–300

12. Jacod, J., Shiryaev, A.: Limit Theorems for Stochastic Processes. Springer - Verlag, NY (2002)
13. Jensen, D., Getoor, L. In: IJCAI Workshop on Learning Statistical Models from Relational Data. (2003)
14. Chen, J., Palmer, M.S.: Towards robust high performance word sense disambiguation of english verbs using rich linguistic features. In: IJCNLP. (2005) 933–944
15. Rakotomamonjy, A., Bach, F., Canu, S., Grandvalet, Y.: Simplemkl. Journal of Machine Learning Research **9** (2008) 2491–2521
16. Bach, F.: Consistency of the group lasso and multiple kernel learning. Journal of Machine Learning Research **9** (2008) 1179–1225
17. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines. (2001) Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.