Online Feature Selection using Grafting

Simon Perkins S.PERKINS@LANL.GOV
James Theiler JT@LANL.GOV
Los Alamos National Laboratory, Space and Remote Sensing Sciences, Los Alamos, NM 87545 USA

Abstract

In the standard feature selection problem, we are given a fixed set of candidate features for use in a learning problem, and must select a subset that will be used to train a model that is "as good as possible" according to some criterion. In this paper, we present an interesting and useful variant, the *online feature selection* problem, in which, instead of all features being available from the start, features arrive one at a time. The learner's task is to select a subset of features and return a corresponding model at each time step which is as good as possible given the features seen so far. We argue that existing feature selection methods do not perform well in this scenario, and describe a promising alternative method, based on a stagewise gradient descent technique which we call grafting.

1. Introduction

In the classic formulation of the feature selection problem, a learning system is presented with a training set $\mathcal{D}$ consisting of $(\mathbf{x}, y)$ pairs, where the $\mathbf{x}$ values are represented by fixed-length numeric feature vectors, and the y values are typically numeric scalars. The learner's task is to select a subset of the elements of $\mathbf{x}$ that can be used to derive a mapping function f from $\mathbf{x}$ to y that is as "good as possible" according to some criterion C, and sparse with respect to $\mathbf{x}$.

This standard formulation assumes that all candidate features are available from the beginning, but consider how things change if features are instead only available one at a time. Assume that we cannot afford to wait until all features have arrived before learning begins, and so the problem is to derive an $\mathbf{x} \rightarrow y$ mapping at each time step, that is as good as possible using a subset of just the features seen so far. We call this scenario *online feature selection* or OFS. The OFS problem is in some sense a *dual* to the standard online learning (SOL) problem. In SOL, the length n of the training feature vectors is fixed, but the number m of training examples increases over time. In OFS, the number of training examples is fixed, but the length of the feature vectors increases over time.

One approach to OFS is simply to take the set of all features seen at each time step, and then apply whatever standard feature selection technique we like, starting afresh each time. However, given that the set of features only increases by one every time step, this is very inefficient. The analogy in SOL would be to retrain from scratch every time a new training example arrived. Therefore we insist that whatever method we use, it must allow efficient incremental updates. Specifically, in a true online situation, we usually have a fixed, limited amount of computational time available in between each feature arrival, and so we want to use a method whose update time does not increase without limit as more features are seen.

Standard feature selection methods can be broadly divided into filter and wrapper methods (Kohavi & John, 1997). How do these approaches adapt to an online scenario?

Filter methods typically use some kind of heuristic to estimate the relative importance of different features. They can be divided into two groups: those that assess the worth of a set of features used together, e.g. Hall (2000), and those that evaluate each feature independently, e.g. Kira and Rendell (1992). We can reject the former group for OFS because the time taken to apply the filter would almost certainly increase as more features are seen. In the current work we also reject the second group because we explicitly want to handle situations where features may only be useful in conjunction with other features.

Wrapper methods directly evaluate the performance of

Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington DC, 2003.

a subset of features by measuring the performance of a model trained on that subset. Under OFS, at each time step we need to consider the possibility not only of selecting the most recently arrived feature, but also of dropping any of the currently selected features. We may also ask if any previously rejected features should now be included. A wrapper approach to answering these questions would require many model retrainings at each update step, and so we reject wrapper methods due to online time constraints.

2. OFS Scenarios

Before we introduce our proposed alternative, it is worth taking a little time to consider under what circumstances OFS is of practical use.

2.1. When Features are Expensive

Most learning systems assume that all the features associated with the training data are ready and available at the start of the learning process. In doing so, they ignore the often considerable computational cost involved in generating those features.

Consider a texture-based image segmentation problem. The task is to assign a label to each pixel in the image according to the texture type that that pixel lies within. Texture is a property of a pixel's neighborhood, so imagine that we have a large number of different "texture filters" that can be applied to each neighborhood in the image, in order to generate features for the pixel. A training image for this task might easily contain tens of thousands of labeled pixels, and each filter might be costly to apply. Rather than spend a lot of computational effort generating all those features up front, it might be far more preferable to generate the features one at a time, and use an OFS learning system to return to the user a model that is as good as possible, given the features seen so far. As time goes on, more and more features are generated, and the model will become better and better.

2.2. Subset Selection in Infinite Feature Spaces

Consider the texture segmentation task again. Now, imagine that we dramatically increase the number of different texture filters that are considered — it is easy to do so by considering different scales, spatial frequencies and texture models. It may well be the case now that there are far more features than we can ever afford to generate in a reasonable time. We are going to have to settle for a solution that depends on only a subset of available features, and we have to pick a reasonable subset without generating all the features first! How is that possible?

One way of managing this situation is to generate features, one at a time in a random order, and to use OFS to select a "best so far" set of features. As time goes on, the currently selected subset of features and associated model will get better and better. When the performance of the model reaches a certain threshold, we can stop generating features and return the latest model.

An intriguing variant of this approach is to use the set of currently selected features to heuristically drive the choice of what candidate features to generate next. For instance we might choose to generate new features that are variations on existing selected features. Perkins et al. (2001) describe an image segmentation system that works along these lines, generating spatio-spectral features that are then combined using a support vector machine.

3. A Framework for OFS

We now turn our attention to developing a formalism and framework for online feature selection.

3.1. Regularized Risk Minimization

In recent years, a lot of attention has been given to the idea that certain forms of *regularization* may be used as an alternative to feature subset selection. This provides the foundation of our incremental approach. To develop the argument, we begin by considering the problem of deriving a good mapping, given a full set of features, as one of *regularized risk minimization*. That is, the criterion to be optimized, C, takes the form:

$$C = \frac{1}{m}\sum_{i=1}^{m} L(y_i, f_i) + \Omega(f) \qquad (1)$$

where $L(\cdot,\cdot)$ is a loss function, and $\Omega(f)$ is a regularization term that penalizes complex mapping functions. We have used f_i as a shorthand for $f(\mathbf{x}_i)$.

3.2. Loss Functions

Different loss functions are appropriate for different types of learning problem. In this paper we will deal with binary classification problems, with y taking values of ± 1, and so a suitable loss function is the binomial negative log-likelihood, used in logistic regression (Hastie et al., 2001, ch. 4):

$$L_{bnll} = \ln(1 + e^{-yf(\mathbf{x})})$$

The BNLL loss function has several attractive properties. It is derived from a model that treats $f(\mathbf{x})$ as the log of the ratio of the probability that $y = +1$ to the probability that $y = -1$, which allows us to calculate[1] $p(y = +1 \mid \mathbf{x})$ using the following relation:

$$p(y = +1 \mid \mathbf{x}) = \frac{e^{f(\mathbf{x})}}{1 + e^{f(\mathbf{x})}}$$

The loss function is also convex in $f(\mathbf{x})$, which has positive implications for finding a global optimum of C. Finally, it only linearly penalizes extreme outliers, which is important for robustness. We denote the mean loss over all training points as $\overline{L}_{bnll}$.

Most of what follows in this paper applies to other commonly used loss functions as well, and we indicate this by dropping the BNLL subscript, except where we need to be specific. A regression task, for instance, would be more likely to employ a sum of squared errors loss function.

3.3. Regularizers

The choice of regularizer in (1) depends upon the class of models used for f. Here, we will restrict ourselves to classes of models whose dependence on $\mathbf{x}$ is parameterized by a weight vector $\mathbf{w}$. Linear models fall into this category, as do various kinds of multi-layer perceptrons and radial basis function networks. A commonly used regularizer for these models is based on a norm of the weight vector:

$$\Omega_p(\mathbf{w}) = \lambda \sum_{j=1}^{n} |w_j|^p$$

where λ is a regularization coefficient, p is a non-negative real number, and n is the length of $\mathbf{w}$. This type of regularizer is the familiar Minkowski ℓ_p norm raised to the p'th power, and so is usually called an ℓ_p regularizer. If $p = 2$, then the regularizer is equivalent to that used in ridge-regression (Hoerl & Kennard, 1970) and support vector machines (Boser et al., 1992). If $p = 1$, then the regularizer is the "lasso" (Tibshirani, 1994). If $p \to 0$ then it counts the number of non-zero elements of $\mathbf{w}$.

The $p = 1$ lasso regularizer has some interesting properties. Firstly, it is the smallest p for which Ω_p is a convex function of $\mathbf{w}$. This means that, if the loss function in (1) is also a convex function of weights, then optimizing C with respect to $\mathbf{w}$ using gradient descent is guaranteed to find the global optimum, since the sum of two convex functions is also convex.

For our work, the second crucial property[2] of the ℓ_1 regularizer is that there is a discontinuity in its gradient with respect to w_j at $w_j = 0$, which tends to force a subset of elements of $\mathbf{w}$ to be exactly zero at the optimum of C (Tibshirani, 1994), which is precisely what we require for a model that is sparse in features. For these reasons we use the ℓ_1 regularizer in our work here.

Note that the model for f may have additional parameters, e.g. bias terms, which we do not include in the regularization.

With the BNLL loss function and ℓ_1 regularization, the learning optimization criterion becomes:

$$C = \frac{1}{m} \sum_{i=1}^{m} \ln(1 + e^{-y_i f(\mathbf{x}_i)}) + \lambda \sum_{j=1}^{n} |w_j| \qquad (2)$$

3.4. Normalization

The Ω_p regularizer penalizes all weights in the model uniformly. This only makes sense if all the features used as input to the model have a similar scale, which can be achieved by normalizing all features as they arrive. A convenient and efficient normalization process is to linearly rescale each feature so that the mean of each feature (over all training data) is zero, and the standard deviation is one, i.e. we rescale incoming feature values x_j to normalized feature values x'_j, using the relation:

$$x'_j = \frac{x_j - \overline{x_j}}{\sigma_{x_j}}$$

where $\overline{x_j}$ is the mean raw feature value, and σ_{x_j} is the standard deviation. It is obviously necessary to use the same rescaling when applying the learned model to new unseen data.

4. Grafting

Perkins et al. (2003) describe a stagewise gradient descent approach to feature selection in a regularized risk framework, called *grafting*.[3] The basic grafting technique is used to build a sparse model from a large set of pre-calculated features, but the same idea can be adapted to OFS, where the features arrive one at a time.

[1] Insofar as $f(\mathbf{x})$ *is* a good model of these log odds, anyway.

[2] In fact, this second property holds for all $p < 2$.

[3] The name is derived from "gradient feature testing".

Grafting is related to other stagewise modeling methods such as additive logistic regression (Friedman et al., 2000), boosting (Freund & Schapire, 1996) and matching pursuit (Mallat & Zhang, 1993).

4.1. Basic Approach

Grafting is a general purpose technique that can work with a variety of models that are parameterized by a weight vector $\mathbf{w}$, subject to ℓ_1 regularization, and other un-regularized parameters, as described in section 3.3 above. We also require that the output of the model be differentiable with respect to all model parameters. The basis for grafting is the observation that incorporating a feature into an existing model involves adding one or more non-zero weights to that model's weight vector. Every non-zero weight w_j added to the model incurs a regularizer penalty of $\lambda|w_j|$. Therefore, it can only make sense to add that weight to the model if the reduction in the mean loss $\overline{L}$ outweighs the regularizer penalty. More specifically, careful examination of (1) and (2) reveals that gradient descent will only take w_j away from zero if:

$$\left|\frac{\partial \overline{L}}{\partial w_j}\right| > \lambda$$

Figure 1 illustrates the criterion graphically. The grafting procedure consists of carrying out this gradient test for each weight that might be added to a model, associated with a newly seen feature. If no weights pass the test, then the corresponding feature is discarded. If at least one weight passes the test, then the weight with the highest magnitude $\partial\overline{L}/\partial w_j$ is added to the model and the model is optimized with respect to *all* its parameters. The tests are then repeated for all the weights that were just tested, since the results may change after optimizing the model.

It is instructive to break the loss derivative into pieces using the chain rule:

$$\frac{\partial \overline{L}}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial L}{\partial f_i}\frac{\partial f_i}{\partial w_j}$$

This is equivalent (apart from the factor of $1/m$) to a dot product in an m-dimensional function space between a loss gradient vector $\nabla_f L$, and a function gradient vector. The loss gradient vector depends only upon the loss function and the current output values of the model, but not on the details of the model. The function gradient depends only upon the details of the model. It is only necessary to calculate the loss gradient vector once in between re-optimizations of the model. This is the key to efficient updates in OFS using grafting: testing to see whether a weight associated with a feature should be added to an existing model simply involves computing a single dot product.

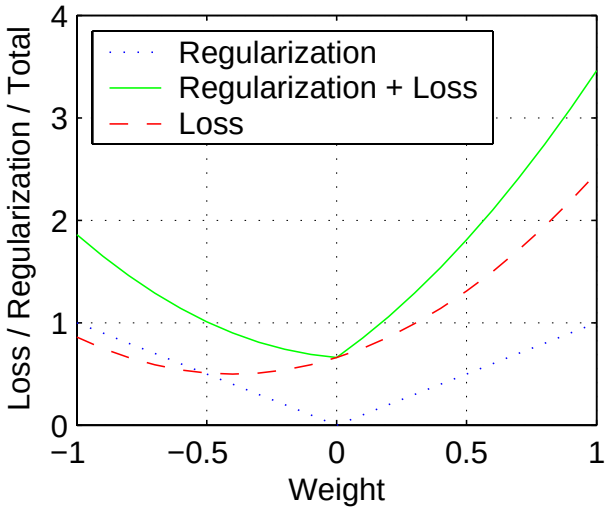

Figure 1. Necessary conditions for progress when adding a weight to the existing model. Downhill progress can only be made if the magnitude of the slope $\partial\overline{L}/\partial w_j$ of the mean loss with respect to the weight at $w_j = 0$, exceeds the slope of the regularizer with respect to the weight, which is $\pm\lambda$. In this case the conditions are not met: the loss term could be reduced with a non-zero weight, but the increase in the regularizer term would more than offset this.

It is also clear from this picture that the magnitude of the dot product will be maximized when the loss gradient and the function gradient line up as much as possible. For the BNLL loss function described earlier, we have:

$$\frac{\partial L_{bnll}}{\partial f_i} = -\frac{y_i}{1+e^{y_i f_i}}$$

4.2. Optimization

Optimization of the model with respect to its parameters can be carried out using any standard unconstrained optimization algorithm. We currently use a conjugate gradient (CG) procedure, on account of its simplicity and low book-keeping overhead. See Fletcher (1987) for implementation details. The CG method requires the use of a "black-box" line minimization method but apart from that the code is very simple.

Before adding any weights to the model, we perform a one-time optimization with respect to the unregularized parameters. After each weight is added,

the model is optimized with respect to all parameters, which may result in certain weights going to zero. In practice care must be taken to catch those weights which go to zero and explicitly prune them, since the gradient discontinuity can cause problems for the line minimization routine.

If the output of the model is a linear function of the model parameters, and the loss function is a convex function of the model output values, then the mean loss is a convex function of the model parameters. All the models and loss functions described in this paper meet these criteria. Since the ℓ_1 regularizer term is also a convex function of model parameters, then these conditions imply that C has a single global optimum with respect to the model parameters. The question arises: how close is the solution found by grafting to this optimal solution?

The grafting solution is at a global optimum with respect to those weights included in the model, since we do a full re-optimization at each step. However, the algorithm described so far does not necessarily lead to the same global optimum that would be found by doing a full optimization including all possible weights and features seen so far. In order to make the correspondence complete, we must ensure that anytime we add a feature to the model, we also go back and reapply the gradient test to all features seen in previous time steps. Although this procedure results in an update time that increases indefinitely as more features are seen, the time taken to test previous features is usually very small compared to the time taken to add a new feature, due to the speed of the gradient test. If necessary, we can impose a limit on how many times a feature can be considered and rejected, before it is removed from future consideration altogether.

4.3. Model Examples

The precise details of the grafting process depend upon the form of the model for f, so we will illustrate grafting for OFS with two example model classes.

4.3.1. Linear Model

Consider a linear model in n features, parameterized by n weights and a bias term:

$$f(\mathbf{x}) = \sum_{j=1}^{n} w_j x_j + b$$

We initialize things by setting $n = 0$, and performing a simple 1-D optimization of C with respect to b.

At each time step t, a new feature arrives in the form of a length m vector:

$$\mathbf{x}_{(t)} = (x_{1,t}, x_{2,t}, \ldots, x_{m,t})^T$$

where $x_{i,t}$ is the t'th feature for the i'th data point. We temporarily augment the existing model with a new weight w_t associated with the new feature. The derivative of the mean loss with respect to w_t is:

$$\begin{aligned} \frac{\partial \overline{L}}{\partial w_t} &= \frac{1}{m} \sum_{i=1}^{m} \frac{\partial L}{\partial f_i} x_{i,t} \\ &= \frac{1}{m} \nabla_f L \cdot \mathbf{x}_{(t)} \end{aligned}$$

If $\left|\partial \overline{L} / \partial w_t\right| > \lambda$, then the weight and corresponding feature are retained in the model, n is incremented, and we optimize with respect to $\mathbf{w}$ and b. Otherwise the weight is dropped and the feature is rejected.

4.3.2. Non-Linear Model

Various non-linear models could be used for OFS and grafting. We use a simple model inspired by additive logistic regression (Hastie et al., 2001) and radial basis function networks:

$$f(\mathbf{x}) = \sum_{j=1}^{n} \left(\sum_{k=1}^{K_j} w_{j,k}\, g\left(x_j - c_{j,k}\right) \right) + b$$

where $g(\cdot)$ can be an arbitrary non-linear 1-D function, but is typically a Gaussian: $g(x) \equiv \frac{1}{\sigma} e^{-(x/\sigma)^2}$. This model is a simple sum of non-linear 1-D functions, each of which is composed of a linear mixture of radial basis functions. For each feature, the non-linear mixture can be composed of between 1 and K_{max} RBFs, where K_{max} is typically 10. The manner of choosing these RBFs and their centers $c_{j,k}$ is detailed below.

We start as with the linear model, setting $n = 0$, and optimizing with respect to the bias term b.

At each time step t, a new feature arrives. This time, instead of considering a single weight associated with the new feature, we consider K_{max} of them, corresponding to the weights on K_{max} different 1-D RBFs. The centers of these RBFs, $c_{t,1}$ through $c_{t,K_{max}}$, are determined by partially sorting the data points according to the value of the t'th feature, in order to find the boundaries of $K_{max} - 1$ equi-percentiles. An RBF center is placed at each of these boundaries, and at the minimum and maximum values. Note that the positions of these centers are fixed by the data, and are not adjustable parameters.

We then proceed in the familiar grafting fashion by calculating derivatives for each of the K_{max} candidate weights:

$$\frac{\partial \overline{L}}{\partial w_{j,k}} = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial L}{\partial f_i} g(x_{i,j} - c_{j,k})$$

and comparing the magnitudes of these derivatives with λ. If none of the derivative magnitudes exceed λ then the feature and corresponding weights are dropped from the model. If at least one derivative magnitude exceeds λ, then we incorporate the weight corresponding the maximum magnitude derivative into the model, and optimize with respect to all existing model parameters. The testing process is then repeated for the remaining weights until they have either all been added, or have all been rejected.

5. Experiments and Results

5.1. The Datasets

We used three datasets in these experiments, labeled **A** through **C**. Each dataset consists of a training set and a test set. Datasets **A** and **B** are synthetic problems, while dataset **C** is a real world problem, taken from the online UCI Machine Learning Repository (Blake & Merz, 1998).

The two synthetic problems are variations of the *threshold max* (TM) problem (Perkins et al., 2003). In the most basic version of this problem, the feature space contains n_r informative features, each of which is uniformly distributed between -1 and +1. The output label y for a data point $\mathbf{x}$ is defined as:

$$y = \begin{cases} +1 & \text{if } [\max x_i] > 2^{(1-1/n_r))} - 1 \\ \text{-1} & \text{Otherwise} \end{cases}$$

The $y = -1$ points occupy a large hypercube wedged into one corner of the larger hypercube containing all the points. The $y = +1$ points fill the remaining space. The constant in the above expression is chosen so that half the feature space belongs to each class. Variations of this basic problem are derived by adding n_i irrelevant features uniformly distributed between -1 and +1, and n_c redundant features which are just copies of the informative features. After generation, the features are ordered randomly.

Dataset **A** is the TM problem with $n_r = 10$, $n_c = 0$ and $n_i = 90$. This dataset explores the effect of irrelevant features in the TM problem.

Dataset **B** is the TM problem, with $n_r = 10$, $n_c = 90$ and $n_i = 0$. This dataset explores the effect of redundant features in the TM problem.

Training and testing sets for each of these problems, each containing 1000 points, were randomly generated. For each experiment involving the synthetic datasets, ten different instantiations of each dataset were generated and the results shown are mean results.

Dataset **C** is the "Multiple Features" database from the UCI repository. This is a handwritten digit recognition task, where digitized images of digits have been represented using 649 features of various types. The task tackled here is to distinguish the digit "0" from all other digits. The training and test sets both consist of 1000 points. The features were all scaled to have zero mean and unit variance before being used here.[4]

5.2. The Experiments

Six different experiments, which we denote by the letters **(a)** through **(f)**, were carried out on each of the three datasets described above:

(a) OFS/grafting with the linear model.

(b) OFS/grafting with the non-linear model.

(c) Step-wise training of a fully-connected version of the linear model.

(d) Step-wise training of a fully-connected version of the RBF model.

(e) Linear SVM applied to all features in batch mode.

(f) Gaussian RBF kernel SVM with default `libsvm` kernel parameters, applied to all features in batch mode.

The grafting algorithms were implemented in Matlab, while the SVM experiments made use of `libsvm` (Chang & Lin, 2001), written in C++. Regularization parameters — λ for the grafting experiments, C for the SVM experiments — were chosen using five-fold cross validation on each of the training sets. The non-linear models used $K_{max} = 10$ and $\sigma = 0.3$.

In order to simulate an OFS scenario, the set of features for each of the datasets was presented to the grafting algorithms one-by-one, in a randomly chosen order.

Experiments **(c)** and **(d)** provide a non-grafting approach to OFS for speed comparison. The models and

[4] All the datasets used in these experiments can be found online at: `http://nis-www.lanl.gov/~simes/data/icml03/`

criteria being optimized correspond exactly to those in experiments **(a)** and **(b)**, but no gradient testing is done to see which weights should be added to the model. Instead, at each time step we simply add all possible new weights to the relevant model before reoptimizing. During the reoptimization process most of the new weights added drop out due to regularization.

5.3. Results and Conclusions

For the OFS experiments **(a)**, **(b)**, **(c)** and **(d)** we recorded the number of weights in the model, the test performance and the elapsed processor time. These measurements are summarized in Figure 2. Since the SVM code we used was implemented in C++ rather than Matlab, a direct timing comparison between batch and online experiments was not performed.

The results show that the stagewise ℓ_1 regularized risk minimization approach is able to select a minimal yet good set of features for the problem at hand, in the presence of many irrelevant or redundant features. The timing experiments demonstrate that grafting is an efficient way of solving the OFS problem, with an update time that is almost independent of the number of features seen so far.

The results also clearly show that the solution obtained is only as good as the underlying model being used. While the non-linear model performed excellently on all problems (outperforming the non-linear SVM in all cases), the linear models performed relatively poorly on the highly non-linear synthetic problems. Despite being fairly flexible, the non-linear model presented here has the property of having a single global optimal solution, which the grafting approach is guaranteed to find.

To summarize, grafting provides an approach to online feature selection that combines the speed of filters with the accuracy of wrappers. The "gradient test" used to decide if a weight should be added to a model, is an extremely quick test, being essentially just a dot product of length m. Yet it gives an exact and direct answer to the question as to whether a given weight should be added to the current model.

References

Blake, C., & Merz, C. (1998). UCI repository of machine learning databases. `www.ics.uci.edu/~mlearn/MLRepository.html`. University of California, Irvine, Dept. of Information and Computer Science.

Boser, B., Guyon, I., & Vapnik, V. (1992). A training algorithm for optimal margin classifiers. *Proc. Fifth Annual Workshop on Computational Learning Theory* (pp. 144–152). Pittsburgh, ACM.

Chang, C., & Lin, C. (2001). LIBSVM: A library for support vector machines. Software available at `http://www.csie.ntu.edu.tw/ cjlin/libsvm`.

Fletcher, R. (1987). *Practical methods of optimization.* Wiley. 2nd edition.

Freund, Y., & Schapire, R. (1996). Experiments with a new boosting algorithm. *Machine Learning: Proc. 13th Int. Conf.* (pp. 148–156). Morgan Kaufmann.

Friedman, J., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: A statistical view of boosting. *Annals of Statistics*, *28*, 337–307.

Hall, M. (2000). Correlation-based feature selection for discrete and numeric class machine learning. *Proc. Int. Conf. Machine Learning* (pp. 359–365). Morgan Kaufmann.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The Elements of Statistical Learning.* Springer.

Hoerl, A., & Kennard, R. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, *12*, 55–67.

Kira, K., & Rendell, L. (1992). A practical approach to feature selection. *Proc. Int. Conf. on Machine Learning* (pp. 249–256). Morgan Kaufmann.

Kohavi, R., & John, G. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, *97*, 273–324.

Mallat, S., & Zhang, Z. (1993). Matching pursuit with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, *41*, 3397–3415.

Perkins, S., Harvey, N. R., Brumby, S. P., & Lacker, K. (2001). Support vector machines for broad area feature classification in remotely sensed images. *Proc. SPIE 4381, Aerosense 2001.* Orlando.

Perkins, S., Lacker, K., & Theiler, J. (2003). Grafting: Fast, incremental feature selection by gradient descent in function space. *Journal of Machine Learning Research.* In press. Also at: http://nis-www.lanl.gov/~simes/pubs.

Tibshirani, R. (1994). *Regression shrinkage and selection via the lasso* (Technical Report). Dept. of Statistics, University of Toronto.

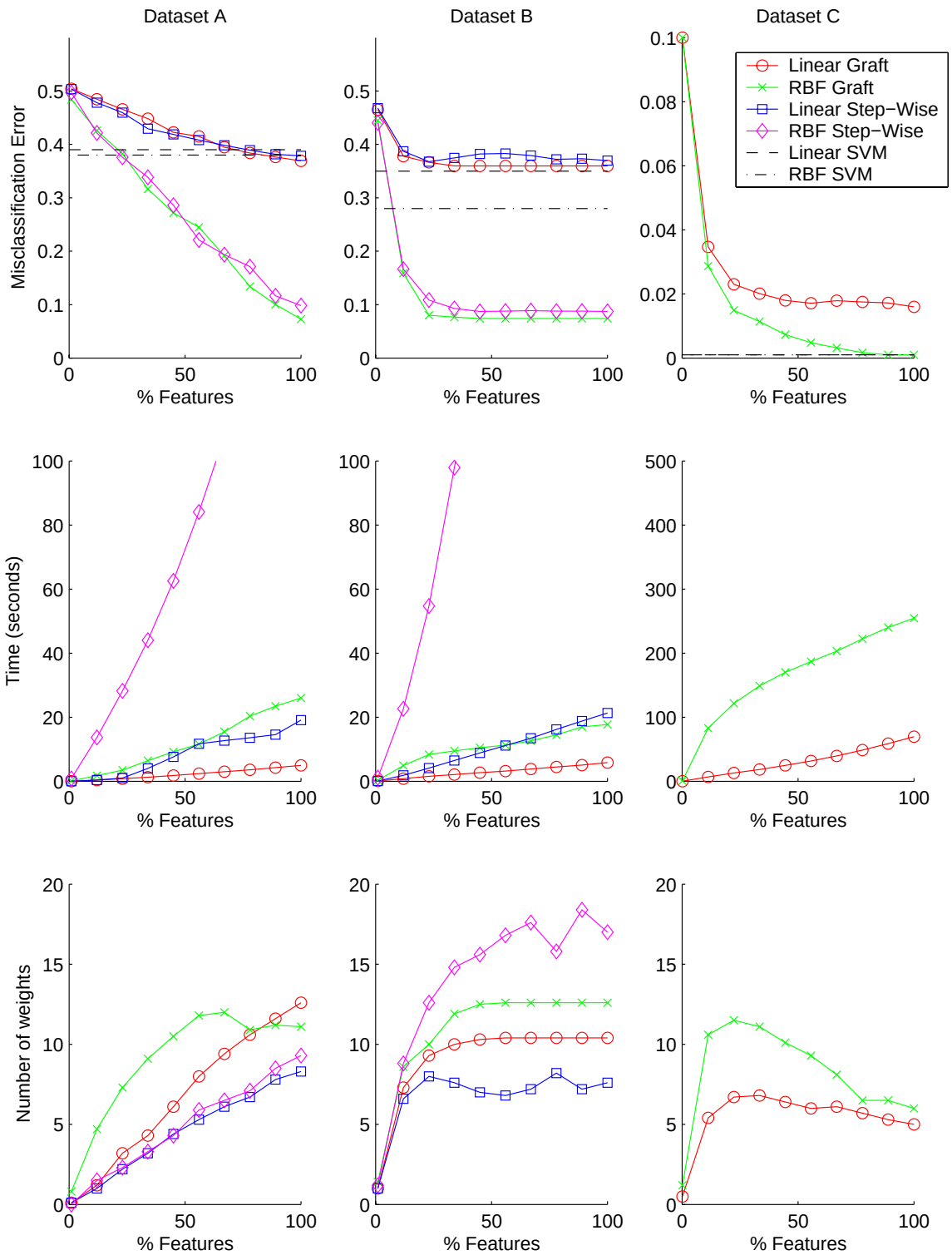

Figure 2. Results of OFS experiments, comparing the greedy and exhaustive versions of grafting with a linear model. Each column of figures relates to one of the three datasets. The graphs show how various measures of the learned model change as the percentage of the total features seen increases. The step-wise experiments were not carried out for Dataset C due to the excessive amount of time required for this method.