The Report Committee for Bryan Arguello
Certifies that this is the approved version of the following report:

A Survey of Feature Selection Methods: Algorithms and Software

APPROVED BY

SUPERVISING COMMITTEE:

Supervisor: Nedialko Dimitrov

Andy Maloney

A Survey of Feature Selection Methods: Algorithms and Software

by

Bryan Arguello, B.S.; M.A.

REPORT

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2015

A Survey of Feature Selection Methods: Algorithms and Software

Bryan Arguello, M.S.E.
The University of Texas at Austin, 2015

Supervisor: Nedialko Dimitrov

The feature selection problem is a major component in disease surveillance since data sources are so costly. This report describes several existing methods for performing feature selection along with software that implements these methods. To help make experimenting with different algorithms easy, we have created a feature selection wrapper package in Python. This wrapper allows the user to easily try different algorithms on the same data set and visualize the results. Experiments are performed to validate that the methods perform as expected.

Table of Contents

Chapter 1

Introduction

1.1 Introduction to the Surety BioEvent App Project

The work described in this master's report is motivated by an influenza surveillance method described by Scarpino *et. al* [11]. There, the underlying problem is feature selection. The state of Texas has a network of healthcare providers, known as ILINet, who provide real-time influenza data to the state and the U.S. Centers for Disease Control and Prevention (CDC). Typically a year after receiving the ILINet data, the CDC receives hospitalization data that describes the number of patients hospitalized due to influenza across the state. Because CDC does not have hospitalization data for the current year, a key public health goal is to estimate that missing hospitalization data from data available in real time — the ILINet data. Historical hospitalization data can be used to select the most effective set of health care providers to include in ILINet, where the effectiveness of a set is measured by its accuracy in predicting the missing hospitalization data.

Though motivated by influenza, the main purpose of this report is to detail feature selection. This report describes the feature selection problem and existing solution methods. In addition, the report gives documentation for Python software to ease further development of feature selection.

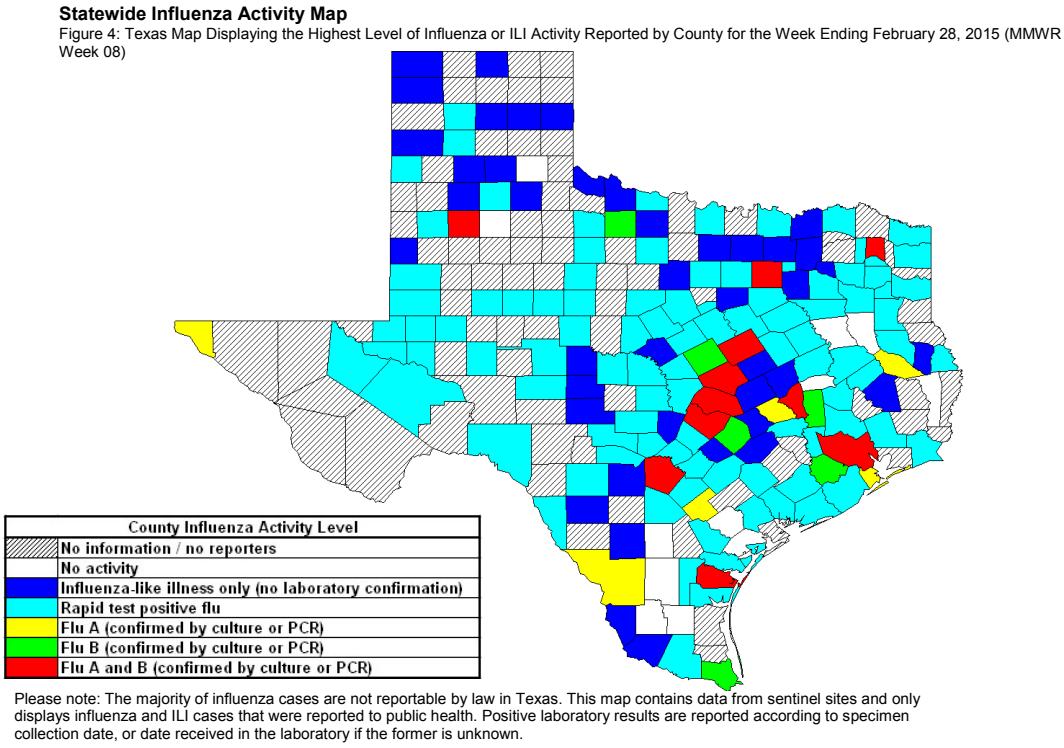

Figure 1.1: This map illustrates influenza activity reported by the Texas ILINet. ILINet is a group of primary healthcare providers that report to the state the number of patients with influenza-like-illness symptoms that visit their office on a weekly basis [1].

1.2 Feature Selection Problem in Influenza Surveillance

To predict missing hospitalization data, there are many data sources of potential use. However, using all data sources may be expensive, or may over-fit the data. The problem of choosing the most informative data sources is called the *feature selection problem*. Feature selection is also referred to as subset selection, feature selection, variable selection, and attribute selection in various fields. Applying feature

selection to ILINet aims to produce the most informative surveillance system while maintaining the number of participating doctors low. Similar feature selection methods can be applied to design surveillance systems for specific geographic regions, a variety of diseases, a variety of prediction goals besides missing hospitalization data, and to select data sources other than primary healthcare providers. The generality of these methods allows them to extend to electronic data sources such as Google searches, Twitter tweets, and WordPress blogs. The government spends millions of dollars funding influenza surveillance systems [2]. Feature selection is an integral part of this surveillance effort.

1.3 Feature Selection

Feature selection is applicable to essentially any prediction problem. The issues that occur in influenza surveillance are common to many applications. As an abstract example to illustrate feature selection, let $x_{1,i} \sim \mathcal{N}(0,1)$ for $i = 1, \ldots, n$. Also let $y_i = x_{1,i}$, $x_{2,i} = x_{1,i}$, and $x_{3,i} \sim \mathcal{N}(0,1)$ and independent from $x_{1,i}$ for $i = 1, \ldots, n$.

To construct a predictor of y from the x's, one may solve the following optimization problem:

$$\min_{\alpha,\beta,\gamma} \|y - (\alpha x_{1,\cdot} + \beta x_{2,\cdot} + \gamma x_{3,\cdot})\|_2^2.$$

Here $\alpha = 1$, $\beta = 0$, and $\gamma = 0$ provides a perfect fit. However $\alpha = \frac{1}{2}$, $\beta = \frac{1}{2}$, and $\gamma = 0$ would provide the same perfect fit. Because obtaining the $x_{2,\cdot}$ data may be resource intensive, we prefer the former solution. Furthermore, because $x_{3,\cdot}$ provides no additional predictive information, there is no reason to choose it as a feature in

prediction. Feature selection identifies $x_{2,\cdot}$ and $x_{3,\cdot}$ as unnecessary data sources.

The purpose of this document is to describe existing methods for feature selection. Chapter 2 describes existing feature selection methods including greedy algorithms, optimization-based methods, and cross validation-based methods. Chapter 3 then outlines existing computational tools in R and Python which perform these feature selection methods. Chapter 4 describes a feature selection wrapper I have written in python to use easily use different feature selection algorithm. Chapter 5 illustrates how to use this wrapper through simple toy examples.

Chapter 2

Existing Methods

This chapter summarizes popular feature selection methods. Section 1 includes forward and backward selection algorithms where single features are added or removed at each iteration. Section 2 describes a simple random selection which can be used to guarantee optimality up to a desired percentage. Section 3 details how to build cross validation into stepwise selection algorithms. Sections 4 and 5 outline algorithms which work by penalizing non-zero regression coefficients. Section 6 concludes with a discussion of proximity methods and orthogonal matching pursuit.

2.1 Stepwise Selection

For the purposes of describing stepwise selection, we introduce the following notation:

indices:

$i \in P$	data sources from which to choose
$j \in \{1, \ldots, n\}$	data index

Data:

P_i	data for data source i
$P_{i,j}$	data point j for data source i
G	target data
G_j	data point j for target data

k number of data sources to choose

Decision Variable:

$S \subset P$ subset of data sources chosen

Model:

$$\begin{aligned} &\max_{S \subset P} R^2(G, S) \\ &\text{s.t.} \quad |S| = k, \end{aligned} \tag{2.1}$$

where α_i are the regression coefficients from fitting G using the P_i's.

The objective function, $R^2(G, S) = \frac{\text{Var}(G) - \text{Var}(G - \sum_{i \in S} \alpha_i P_i)}{\text{Var}(G)}$, is just one potential measure of the prediction accuracy. Of course, one could substitute other measures, but for the purposes of simplicity we use R^2 to describe the method. One advantage of R^2 over other measures is that it is submodular over the set of subsets of P [5].

A candidate algorithm for a subset selection problem with a submodular objective function is forward selection. Nemhauser proves that, under reasonable assumptions, the greedy forward selection algorithm is a good approximation to the optimal solution when the objective function is submodular [10].

Algorithm 1 Forward Selection Algorithm

```
1: function FORWARD SELECTION(k, P)
2:     Set S = ∅
3:     while |S| < k do
4:         Set e = argmax_{e∈P−S} R²(G, S ∪ {e})
5:         Set S = S ∪ {e}
       return S
```

Intuitively, forward selection works because a submodular function exhibits

the marginal returns property for adding single features.

A similar greedy algorithm which works by removing single features is backward elimination:

Algorithm 2 Backward Selection Algorithm

```
1: function BACKWARD SELECTION(k,P)
2:     Set S = P
3:
4:     while |S| > k do
5:         Set e = argmax_{e∈S} R²(G, S − {e})
6:         Set S = S − {e}
       return S
```

Even though backward elimination is also a popular method, it does not have the same approximation guarantees as forward selection, even when the objective function is submodular.

2.2 Random Selection

Since the feature selection problem has $\binom{P}{k}$ solutions, exhaustive search is inefficient. However, if we choose a sufficiently large number of subsets then we are guaranteed to be in a desired upper percentile of solutions with a reasonably large probability. More specifically, let α be a desired percentile and n be a number of samples subsets. Then the probability that one of those subsets is in the upper α percentile is

$$p = 1 - (1 - \alpha)^n \tag{2.2}$$

For example, we can be in the top ten percentile with a probability of 0.995 with just 50 samples. The random selection algorithm is

Algorithm 3 Random Selection Algorithm

```
1: function RANDOM SELECTION(α, p)
2:     Determine n
3:     for i = 1...n do
4:         Choose a random subset S with |S| = k
5:         if R²(G, S) > R²(G, S_best) then
6:             Replace S_best with S
       return S_best
```

Even though random selection is not efficient at finding the optimal solution, it guarantees a probabilistic solution quality regardless of the structure of the objective function.

2.3 Cross-Validation Based Stepwise Selection

Forward selection proves to be relatively efficient for solving equation (2.1). However, forward selection alone does not address the problem of overfitting. Since linear regression is done using every data point, as we add more and more predictors, we will get a better and better fit on those points. However, when testing on a never-before seen data point, the fit may be very bad. Using cross-validation within an optimization scheme can help to avoid overfitting.

Using the same setup as in forward regression, a cross validation-based forward selection algorithm is

Algorithm 4 Forward Selection with Cross-Validation

function FORWARD SELECTION CV(k, P)
    Store 20% of all data to validate that the algorithm is effective at the end
    Set $S = \emptyset$ and set N to a large number for sampling
    while $|S| < k$ **do**
        for $e \in P - S$ **do**
            Set $T = S \cup \{e\}$
            for i **do** = 1...N:
                Randomly split the remaining 80% of data into a set D with

                30% of the data and a set E with 70% of the data

                Do a least squares fit of the data sources in T to G using E

                Using the coefficients from the fit, create a predictor

                Calculate the mean-squared-error(MSE) between the predictor and G on D

            Let m be the average of the MSE values. Use that value in place of R^2 as an objective function value
        Choose the e which gave the largest m value and let $S = S \cup \{e\}$
    Do a least square fit using S and the 20% that was stored. If the MSE is reasonably small, then consider this algorithm successful
return S

Though this algorithm helps to avoid overfitting, it has less theoretical justification than both random selection and stepwise selection. Experiments using all four optimization schemes are performed in Section 5.

In addition to feature selection, cross validation can also be used for model selection. In particular, cross validation can be used to select parameters in any of the methods discussed in 2.4 and 2.5

2.4 Parameter Minimization Based Methods

The methods in this section work by minimizing the least square error plus a penalty on non-zero coefficients. The idea is to drive the coefficient value of data sources that do not provide much benefit to zero. All of these methods have the same general approach. The objective function they minimize is a sum of the squared error, plus a penalty term for non-zero coefficients. The only difference from one method to another is the penalty term they use.

2.4.1 Ridge Regression

One possible penalty term is the L^2 norm of the parameter vector. In optimization, this technique is known as Tikhonov regularization while in statistics, it is known as ridge regression. For a given penalty term weight $\lambda \in \mathbb{R}$, the ridge regression problem is

$$\min_{\beta} \quad \sum_{j=1}^{n} \left(G_j - \sum_{i \in P} \beta_i P_{i,j} \right)^2 + \lambda \|\beta\|_2^2$$

Model (2.4.1) is an unconstrained quadratic convex program, so it has an analytic solution. Even though, by design, ridge regression shrinks the regression coefficients, it does not usually shrink the coefficients all the way down to zero. One may ignore features with small coefficients, but that alters the prediction output. If one performs feature selection by removing the small coefficients, re-adjusting the remaining coefficients through another parameter estimation may be advisable. For more details on ridge regression, see either the original paper on ridge regression [8] or the original paper on LASSO [12].

2.4.2 LASSO

In 1994 Robert Tibshirani developed a modified version of ridge regression, LASSO, which enjoys the benefits of ridge regression while remedying some of its downfalls. The LASSO model is

$$\min_{\beta} \quad \sum_{j=1}^{n} \left(G_j - \sum_{i \in P} \beta_i P_{i,j} \right)^2 + \lambda \|\beta\|_1 \tag{2.3}$$

For a given $\lambda \geq 0$, Model (2.3) is equivalent to

$$\begin{aligned} \min_{\beta} \quad & \sum_{j=1}^{n} \left(G_j - \sum_{i \in P} \beta_i P_{i,j} \right)^2 \\ \text{s.t.} \quad & \|\beta\|_1 \leq t \end{aligned} \tag{2.4}$$

for some $t > 0$. In particular, let $\hat{\beta}$ be a solution to (2.3). Then $t = \frac{1}{\lambda}\|\hat{\beta}\|_1$ is such a t that makes (2.3) and (2.4) equivalent.

Tibshirani demonstrates that both theoretically and in practice, LASSO shrinks coefficients all the way down to zero [12]. Intuitively, this can be understood by comparing the constraint sets of ridge regression and LASSO. The constraint set for ridge regression is a ball, while the constraint set for LASSO is a rotated hypercube. Optimality occurs where a contour of the objective function, an ellipsoid, first intersects the constraint set. Ellipsoids are much more likely to intersect a hypercube at a point on an axis than a ball.

Unlike ridge regression, LASSO does not lend itself to a closed-form analytic solution. Both models are convex optimization problems. However, model (2.4) does

not have a differentiable constraint function. The constraint could be rewritten as $2^{|P|}$ linear constraints. A row generation algorithm for solving the resulting convex problem with an exponential number of constraints is given by Tibshirani [12]. This row generation algorithm is guaranteed to converge in exponential time. However as with many row generation algorithms, it converges more quickly in practice.

A feature of LASSO is that it will only choose one out of a collection of highly correlated features. In addition, it usually does not perform well when the number of data sources available is much larger than the number of observations. LASSO also tends to perform poorly in terms of prediction when the number of data sources with large LASSO coefficients is small [14].

2.4.3 Elastic Net

Nearly a decade after the creation of LASSO, Zou and Hastie developed a similar model designed to mitigate the problems encountered by ridge selection and LASSO. Elastic net, not surprisingly, is a hybrid of ridge regression and LASSO. Zou and Hastie developed elastic net by first introducing the naive elastic net model [14]

$$\min_\beta \sum_{j=1}^{n} \left(G_j - \sum_{i \in P} \beta_i P_{i,} \right)^2 + \lambda_1 \|\beta\|_1 + \lambda_2 \|\beta\|_2. \tag{2.5}$$

They note that this problem is equivalent to

$$\begin{aligned} \min_\beta \quad & \sum_{j=1}^{n} \left(G_j - \sum_{i \in P} \beta_i P_{i,j} \right)^2 \\ \text{s.t.} \quad & \alpha_1 \|\beta\|_1 + \alpha_2 \|\beta\|_2 \leq t \end{aligned} \tag{2.6}$$

where $\alpha_1 = \frac{\lambda_1}{\lambda_1 + \lambda_2}$ and $\alpha_2 = \frac{\lambda_2}{\lambda_1 + \lambda_2}$ for some $t \in \mathbb{R}$.

This model reduces to ridge regression if $\lambda_1 = 0$ and LASSO if $\lambda_2 = 0$. Incidentally,

the naive elastic net model is no more difficult to solve than LASSO. Model (2.5) can be transformed into a model of the same form as model (2.3) through an augmentation and multiplication [8].

Model (2.5) is described as naive since it only predicts well in terms of minimizing MSE on test data when either α_1 or α_2 are close to zero. This is to say that naive elastic net is only good when it is mimicking ridge regression or LASSO. However, after multiplying all components of a naive elastic net solution by $\frac{1}{\sqrt{1+\lambda_2}}$, this is no longer the case. The technique of solving the naive elastic net and then scaling its solution is known as elastic net.

Elastic net performs well in terms of prediction and driving coefficients all the way down to zero. In addition, as opposed to LASSO, it either includes or excludes an entire collection of highly correlated features [14].

2.4.4 Nonnegative Garrote

Around the time that LASSO was developed, Leo Breiman developed a method known as the nonnegative garrote (nng) [4]. This solution method takes an initial solution of β values and performs a linear regression while shrinking data source coefficients all the way down to zero. One candidate for an initial solution is the least square coefficients. Alternatively, one can obtain a ridge regression solution and apply the nng to shrink the small coefficients all the way down to zero.

2.5 Other Methods

2.5.1 Proximity methods

All methods mentioned so far are supervised: they work by optimizing the coefficients β to fit a target G. One may also take a different approach to feature selection. Instead of using a target, some measure of proximity can be used to pick features which are close to one another. Typical proximity measures include the correlation coefficient, mutual information, and pointwise mutual information. These measures tend to choose the data sources which maximize information [7]

2.5.2 Orthogonal Matching Pursuit

The orthogonal matching pursuit (OMP) is a greedy algorithm originally introduced for sparse representation of a signal. OMP views both the goal G and the data sources P_i as long vectors, with length equal to the number of observations. The algorithm works projecting G onto each of the P_i vectors, and picking the P_i that gives the longest projection. It then subtracts that projection from G, calls the result the residual G, and repeats the process with by projecting the residual G onto the remaining P_i. See Tropp for more details on using OMP to approximate a signal [13].

Chapter 3

Existing Software

3.1 Scikit Learn and R

There are several existing software solutions that implement the feature selection algorithms mentioned in Chapter 2. Two popular tool sets are Python's Scikit Learn module and R. R has various machine learning algorithms implemented throughout several packages. Both R and Scikit Learn are open source and have a large community continuing to use and develop machine learning tools. For documentation on how to use Scikit Learn's feature selection tools, see the "Feature Selection" and "Generalized Linear Model" sections from the Scikit API page [3].

There is a folklore belief that forward selection does not perform well. This belief is substantiated by lack of a forward selection implementation in both Scikit Learn and R. In addition, the example shown in Figure 5.3 from Chapter 5 illustrates forward selection producing a suboptimal solution. On the same example, all other algorithms mentioned in Chapter 2 produce the optimal solution.

To ease development of feature selection in Python, this report introduces a Python feature selection wrapper. This wrapper is designed to allow a developer to easily implement feature selection on top of any prediction method – whether or not it is one already in Scikit Learn or R, or a custom method. The full software details are given in Chapter 4.

Chapter 4

Custom Feature Selection Wrapper

Generally, there are several components that can be used in feature selection. A predictive function creates a model for a target data source from a set of features. A widely-used example of a predictive function is the linear regressor from least squares regression. An objective function can then be used to compare two models to see which is better. The R^2 function from Chapter 2 is a useful example of an objective function. Finally, an optimization algorithm uses an objective function in order to pick subsets of features based on previous evaluations until the objective function is small enough.

4.1 Wrapper Layout

The custom feature selection wrapper package is designed to separate data, modeling, and optimization components. The first step in using the software is to customize the data import function. Once data is imported, the user has the capability to easily run different optimization algorithms. The feature selection wrapper package has the following directory structure:

```
├── data_retriever.py ..... Used for importing data.
└── __init__.py
```

```
└── optimization
    ├── __init__.py
    ├── problem.py
    ├── objective_functions.py ..... Contains objective functions.
    ├── optimization_algs.py ..... Contains feature selection algorithms.
    └── predictive_functions.py ..... Contains predictive functions.
```

The main module, `problem.py`, contains a class called `FS_problem`, which includes the test data and functions for interacting with the data. A user can import data into an instance of `FS_problem` through the `data_retriever` module and then perform optimization and visualization. A user can then add feature selection algorithms into `optimization_algs.py` and objective functions into `objective_functions.py`. In addition, the user can add predictive functions such as least squares regressors into `predictive_functions.py`.

4.2 Adding Data to an Instance

Once an instance of `FS_problem` is created, a list of data sources is automatically stored in the instance's data variable. A data source must be in the form of a dictionary as

```
{
  "metadata": {
    "name": "data_name"
  },
  "data": {
    "times": [times],
    "values": [values]
  }
}
```

where `data_name` is the name of the data source, and `times` and `values` are parallel lists which give the data for a time series. The code expects all data sources to have associated time series that match in length and time-points. However, data sources may contain additional key-value pairs anywhere in the dictionary structure, but the above core structure is required to be present. For the rest of this report a data source refers to such a dictionary with.

The `data_retriever` module contains a single function called `get_data()` which returns a two-tuple. The first component of this tuple is a list of data sources for use in feature selection, and the second component is the target data source. How to import the data sources and the target is up to the user. Currently, `get_data()` creates the data sources from csv files and prompts the user to choose one as a target. Once a target data source is chosen, it is removed from the list of data sources so that it will not be used in feature selection.

4.3 Setting up a Problem and Performing Feature Selection

Suppose the user creates an instance of `FS_problem` called "problem" with the command

```
import optimization.problem as problem
problem = problem.FS_problem()
```

During instantiation, data is automatically imported into problem through a call to `get_data()`.

The user is now ready to begin performing optimization. Since all methods from Chapter 2 have been implemented in the wrapper, the user can use any of them to perform feature selection. Alternatively, the user can extend our wrapper by adding new optimization algorithms and use those algorithms with their problem instance. This report gives details on how to extend our wrapper in Section 4.4.

4.3.1 Specifying an Objective Function

Suppose the user decides to perform forward selection with R^2 as an objective function. The R^2 objective function has been implemented within `objective_functions.py`, and its docstring is

```
def R_squared(problem, subset, **kwargs):
    """Returns the coefficient of determination R^2

    Description
    -----------
    This statistical quantity tests how well predictive data fits
    model data

    Parameters
    ----------
    :param problem: problem being solved
    :param subset: data for prediction

    :type problem: FS_problem
    :type subset: list of data sources

    :return: R_squared correlation of determination
    :rtype: float
```

```
    Reference
    ---------
    http://en.wikipedia.org/wiki/Coefficient_of_determination
    """
```

Since the name of the R^2 function is `R_squared`, the user would issue the command

```
problem.objective_function = "R_squared"
```

Now any optimization algorithm that requires an objective function will use the `R_squared function`.

4.3.2 Specifying a Predictive Function

Suppose the user wants to use a least squares regressor with `R_squared`. One of the functions within `predictive_functions.py` is

```
def lin_reg(goal_series, series_collection):
    '''Returns a predictive function using least_squares

    Description
    -----------
    Returns Ax* where x* = argmin_{x}( || Ax - b ||_2 )
    The matrix, A, is obtained by putting the lists from series_collection
    into a column of A.  The vector b is simply goal_series
```

```
    Parameters
    ----------
    :param goal_series: the vector b
    :param series_collection: list of column vectors of A

    :type goal_series: list of floats
    :type series_collection: list of list of floats

    :return: least squares coefficients and a preditive time series
    :rtype: (list of floats, numpy.array)
    '''
```

To use this predictive function, the user would issue the command

```
problem.predictive_function = "lin_reg"
```

Now any time a predictive function is required, the `lin_reg` function will be used.

4.3.3 Performing Feature Selection

Forward selection is one of the optimization algorithms implemented in `optimization_algs.py`. Its docstring is

```
    def forward_selection(problem, **kwargs):
        """Forward selection heuristic

        Requirements
```

```
    ------------
    kwargs['choose'] is a positive integer which specifies exactly how many
    data sources to choose in feature selection

    Parameters
    ----------
    :param problem: problem to be solved

    :type problem: FS_problem

    :return: (optimal subset, optimal objective value)
    :rtype: (list of data sources, float)

    Reference
    ---------
    "Optimizing Provider Recruitment for Influenza Surveillance Networks"
    """
```

Since `choose` is a required parameter to `forward_selection`, it will be required as a keyword argument to `optimize()`. To perform feature selection, the user will use the command

```
problem.optimize("forward_selection", choose = 3)
```

The keyword `choose` is given since it is a requirement of forward selection. If there are any requirements for the objective function used by an optimization algorithm, it should be specified in the call to `optimize()` as well. Since the `R_squared` objective function has no requirements, only choose is specified in this call to `optimize()`.

The names of the three data sources chosen by forward selection are then displayed along with a graph containing G and the least squares fit of the three data

sources to G.

```
In [3]: current.optimize('forward_selection', choose = 3)
Optimal solution:
Ebay
Facebook
Twitter

Objective value: 0.93620780146
```

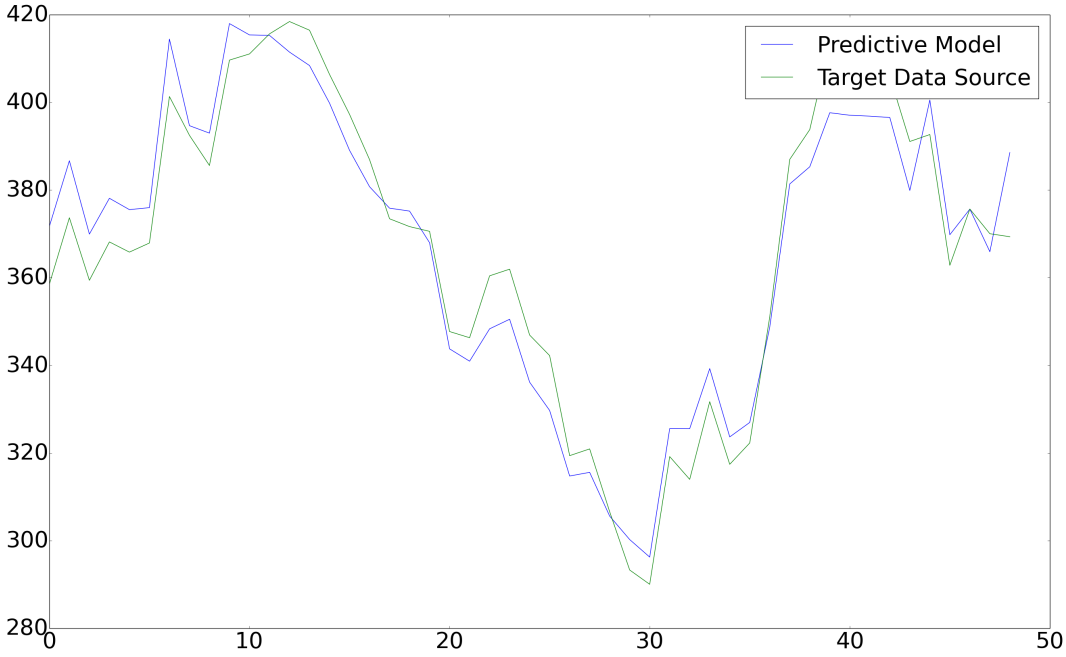

Figure 4.1: The output of a call to `optimize()` using forward selection and `choose` parameter of three. The features chosen by this run are Ebay, Facebook, and Twitter. The target data source is shown in green, and the model created from the chosen data sources is shown in blue.

4.3.4 Switching Feature Selection Algorithms

Suppose the user now wants to see which subset LASSO chooses, as compared to forward selection. The currently-implemented LASSO algorithm uses the Scikit Learn LASSO function, so it is not necessary to specify an objective function or predictive function. However, as is documented in the LASSO docstring, `ridge_reg_coef` must contain a nonnegative float. The user can run LASSO with a command such as

```
problem.optimize("LASSO", LASSO_reg_coef=0.5, coef_tolerance=0.05)
```

The user can continue trying different feature selection algorithms by specifying parameters as keyword arguments as necessary according to the algorithms' docstrings. See Figure 4.2 for the output.

4.4 Extension

A user can add new optimization algorithms into `optimization_algs.py`, new predictive functions into `predictive_functions.py`, and new objective functions into `objective_functions.py`. When adding these, it is important to be explicit in the docstring about the required parameters since the user needs these in order to run the algorithm.

```
In [6]: current.optimize('LASSO', LASSO_reg_coef = 0.5, coef_tolerance = 0.05)
Optimal solution:
Facebook
Twitter
Yelp

Objective value: 0.999989247505
```

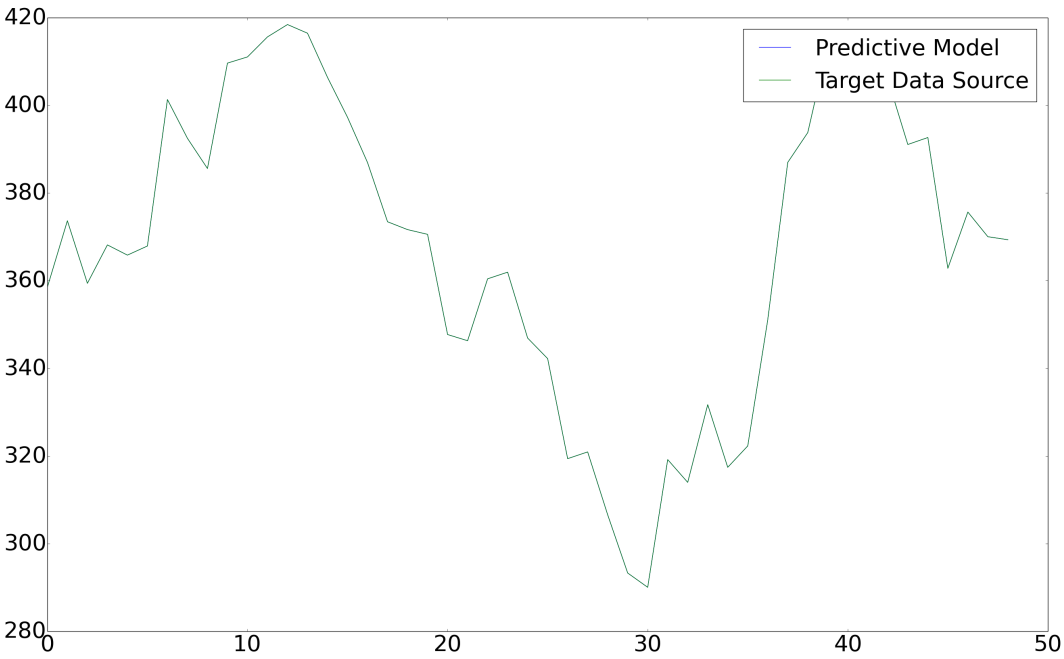

Figure 4.2: The output of a call to `optimize()` using LASSO

Chapter 5

Experiments

5.1 Toy model

In this chapter, we use a small toy example to illustrate how the feature selection wrapper can be used as well as to validate the optimization algorithms. The data sources for this problem are closing stock prices for nine different companies.

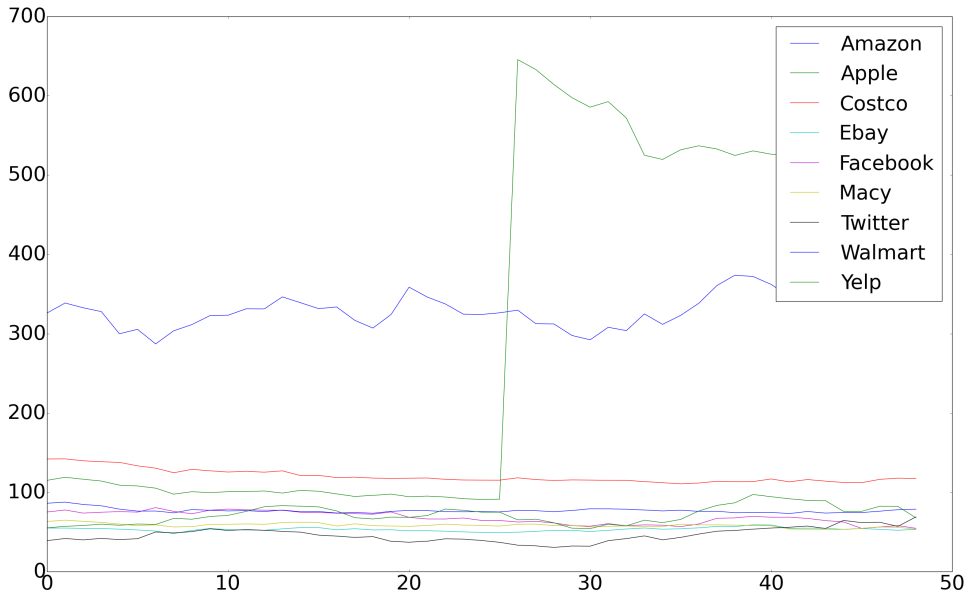

Figure 5.1: The data sources used in this experiment are weekly closing stock prices for December 2014-January of 2014.

One way to test that the implemented optimization algorithms work is to test them with a G that is constructed from a linear combination of some data sources. In this toy example, G is constructed as

$$G_j = \text{Yelp}_j + 2 * \text{Twitter}_j + 3 * \text{Facebook}_j$$

Intuitively, we expect a good feature selection algorithm to choose Yelp, Twitter, and Facebook—assuming that all of the data sources are linearly independent.

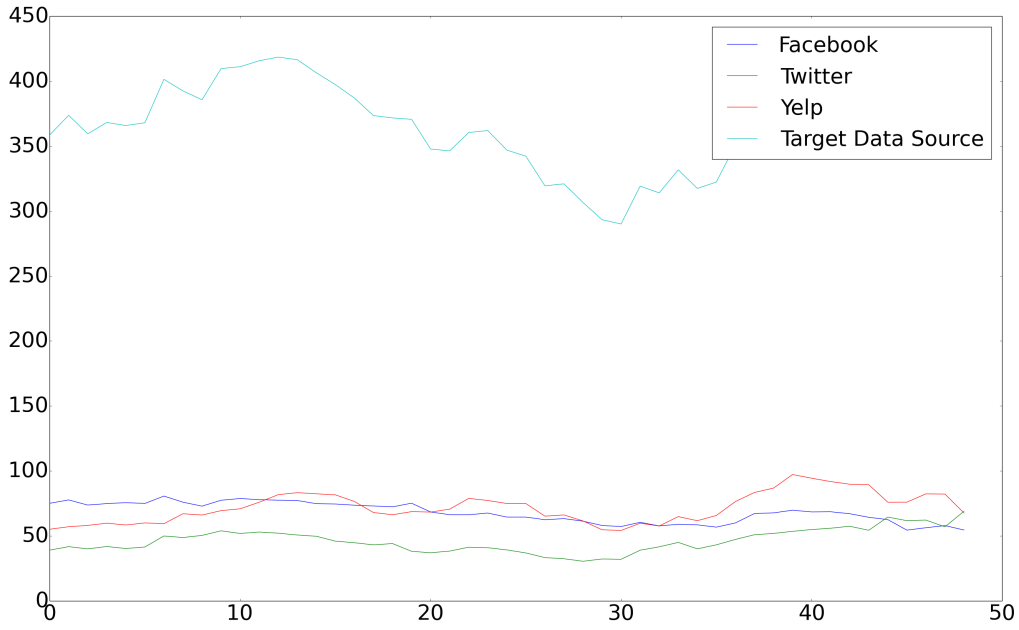

Figure 5.2: The constructed data, G, in light blue, is constructed from a linear combination of Yelp, Twitter, and Facebook

All models discussed in Chapter 2 along with their cross-validation counterparts are implemented in the current version of the feature selection wrapper

All algorithms except for forward selection with an R^2 objective function choose Yelp, Twitter, and Facebook. When the choose parameter for forward selection is set to three, forward selection chooses Ebay, Facebook, and Yelp (see Figure 4.1). However, when the choose parameter is set to four, Ebay, Facebook, Twitter, and Yelp are chosen. This simple example illustrates the benefit of performing feature selection with multiple algorithms, to validate the robustness of the set of output features.

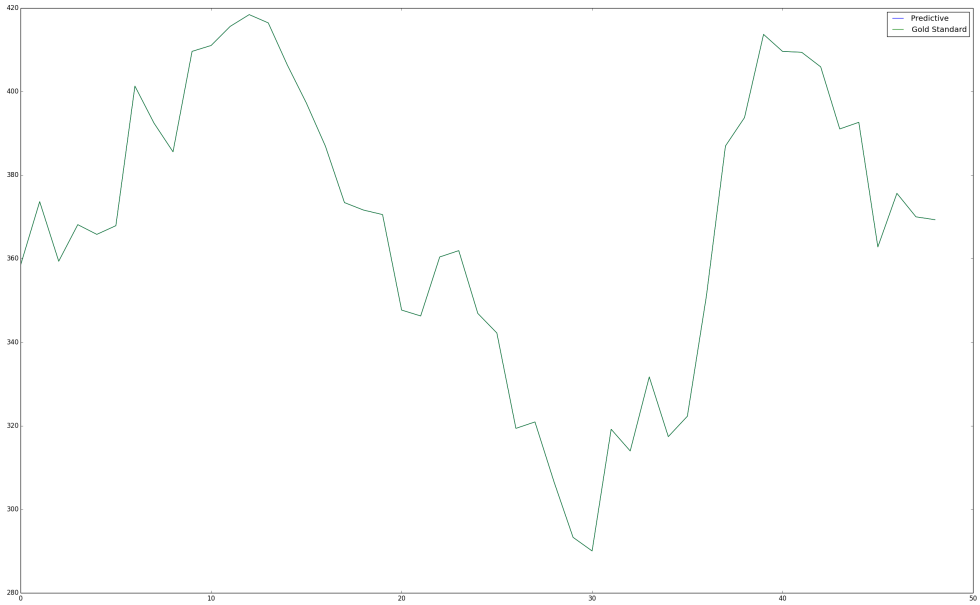

Figure 5.3: Forward selection with an R^2 objective value and a `choose` parameter of four selects Ebay, Facebook, Twitter, and Yelp. This is a perfect fit, yet Yelp, Twitter, and Facebook give a perfect fit

Chapter 6

Discussion

This report surveys only a subset of feature algorithms. More feature selection algorithms can be explored by considering other predictive functions, objective functions, and optimization algorithms. Additional predictive functions to consider are logistic regression and various nonlinear regression types. Various norms and statistical quantities can be used for objective functions as appropriate for applications. Finally, for each feature selection problem, there are usually various applicable optimization algorithms that can be used.

Because Python and R are so popular in the scientific computation community, Scikit Learn and the R packages were the only tools discussed in this report. There are numerous programming languages and computational tools which have their own feature selection libraries. In particular, more low-level language such as C and Java may be better for solving large problems. Additionally, feature selection tools in MATLAB and Mathematica can be used in applications where codes are already written using these tools.

There were several software design decisions made in favor of the application funding this work. Future effort would include providing a better separation of the data from the feature selection wrapper. Users may wish to use some other data structure in place of the dictionary structure imposed by the feature selection wrapper. Additionally, it may be more convenient to replace the strings used to specify objective and predictive functions with actual functions to provide even greater flexibility.

The software developed through this masters report can be used to easily apply many feature selection algorithms to a feature selection problem. The data structure imposed on the user is simple and useful, and extending the software to include additional algorithms has been made simple and transparent. In addition, the user can easily add in extra capabilities since the code is modular and well-documented.

Bibliography

[1] https://www.dshs.state.tx.us/idcu/disease/influenza/surveillance/2013.

[2] http://globalbiodefense.com/2013/09/19/chemical-and-biological-defense-technologies-baa-awards/.

[3] http://scikit-learn.org/stable/modules/classes.html/.

[4] Leo Breiman. Better subset regression using the nonnegative garrote. *Technometrics*, 37(4):373, November 1995.

[5] Abhimanyu Das and David Kempe. Algorithms for subset selection in linear regression. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 45–54. ACM, 2008.

[6] Abhimanyu Das and David Kempe. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. *arXiv preprint arXiv:1102.3975*, 2011.

[7] Isabelle Guyon and AndrÃĺ Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

[8] Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12:55, 1970.

[9] A. Krause and D. Golovin. Submodular function maximization. Technical report, ETH Zurich, 2012.

[10] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1988.

[11] Samuel V. Scarpino, Nedialko B. Dimitrov, and Lauren Ancel Meyers. Optimizing provider recruitment for influenza surveillance networks. *PLoS Comput Biol*, 8:1–12, 2012.

[12] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 58:267–288, 1996.

[13] Joel A. Tropp and Anna C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on Information Theory*, 53(12):4655–4666, December 2007.

[14] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67:301–320, 2005.

Vita

Bryan Arguello was born and raised in Trinidad, CO. After graduating from Trinidad High School, he spent a year taking core math and science courses at Trinidad State Junior College. Having discovered that he enjoyed mathematics, Bryan studied applied mathematics and computational sciences at the Colorado School of Mines (Mines). Part of his education at Mines included an REU in numerical radial basis function methods at Hong Kong and a study abroad with the Budapest Semesters in Mathematics program. After he received his B.S. from Mines, Bryan spent two years at the University of Colorado Boulder studying pure mathematics and obtaining an M.A. in mathematics. In 2013, Bryan started an M.S.E. program in operations research and industrial engineering at the University of Texas at Austin. There he focused on optimization, machine learning, and computational optimization. In addition to his educational career in mathematics and operations research, Bryan has served in various instructional capacities as a tutor, teacher, and faculty member in several high schools, community colleges, and universities continuously from 2004 through 2015. He also spent a year working as a software developer/QA specialist for OpenX. Currently, Bryan works as an operations research modeler at Sandia National Labs.

Email Address: bromero314@gmail.com

This report was typed by the the author.