Feature Selection and Binary Classifier Analysis of Labeled Datasets

Joshua Shapiro, Larisa Sidorovich
Department of Computer Science
The George Washington Unversity
Washington DC

Abstract

The purpose of the project is as follows: given multiple datasets with binary labels, we want to analyze the accuracy of a variety of binary classifiers combined with different feature selection algorithms. The end goal of this analysis is to discover the best feature selector and classifier pair for each dataset, and to see which features are most significant in classification. Additionally, we want to see if similarly structured data can be classified accurately using similar classification and feature selection methods.

1 Introduction

Binary classification and feature selection have been problems long studied with a variety of solutions. We want to analyze the accuracy of a handful of common classification algorithms paired with common feature selection algorithms over multiple datasets. From this analysis we hope to discover the best feature selector and classifier combination for each dataset, and see which features are most significant to classification. Additionally, we want to see if similarly structured data can be classified accurately using similar classification and feature selection methods.
We begin by describing the datasets chosen and what makes them good options for this project. We then cover how the data was preprocessed in preparation for analysis. Then the feature selection algorithms are described in detail, and the most significant features for each dataset are explained. Eventually we move on to an analysis of different classifiers, and finally we discuss the accuracy of the classifiers and feature selection algorithms on each dataset.

2 Data

Given the focus on a variety of feature selection algorithms and classifiers, only two datasets were chosen. This enabled us to spend more time choosing the feature selection algorithms and classifier types we wanted to test. After careful consideration we decided to study the Mushroom Dataset from the UCI Machine Learning Repository [5] and the Voice Dataset from Kaggle [1]. Both datasets have approximately the same number of features and have binary labels, so they seemed like good candidates for testing.
For the mushroom dataset, there are 8124 data points each with 22 attributes. Each data point represents a mushroom and is labelled either edible, poisonous, or unknown. The dataset on the UCI Machine Learning Repository combined poisonous and unknown labels, so the data can be analyzed using a binary classifier. The features are all categorical, relating to attributes such as cap color, odor, gill size, and habitat. Some of the data points are missing feature values, and some feature values are the same for every mushroom [5].
For the reasons stated above, the mushroom data had to be pre-processed. This included removing columns 11 and 17 (stalk-root & veil color) since they were missing data or all had the same values.

Additionally, the categorical values for each attribute were converted to numerical values and then normalized to have values between 0 and 1. After this pre-processing step, the mushroom data consisted of 8,124 binary labeled data points with 20 numerical features each.
The voice data set from Kaggle consists of 3,168 recorded voice samples that were pre-processed by acoustic analysis in R. The data points consist of 20 numerical features related to attributes such as of frequency, average fundamental frequency, interquartile range, and frequency range. Each sample is labeled either male or female [1]. The only required preprocessing was to randomly permute the data points as they were originally sorted by label. This is necessary as we are testing perceptron classifiers among others. By the end of this pre-processing step, we had two binary labeled data sets with 20 numerical features each.

3 Feature Selection

It is important to note that the purpose of this project is to analyze feature selection algorithms, not feature extraction algorithms. While both are forms of dimensionality reduction, we want the results of our project to show us which features are most important in classification, and feature extraction does not allow this. For this reason, dimensionality reduction algorithms that transform the high dimensional space to low dimensional space will not be tested. Below are descriptions of the three feature selection algorithms that we have chosen.

3.1 Sequential Feature Selection Forward

This algorithm is considered a wrapper algorithm as it is dependent on a classifier. It evaluates feature subsets of the original data using cross-validation, and effectively performs a greedy state space search over all possible feature subset permutations. As this algorithm is greedy, it can unfortunately get stuck in local maxima. Additionally, this algorithm is slow to run as the feature subset evaluation function requires cross-validation and classification of the training data. The general algorithm for sequential feature selection forward is as follows [3] :

1. Start with the empty set $Y_0 = \{0\}$
2. Select the next best feature $x^+ = arg\ max_{x \in Y_k} J(Y_k + x)$
3. Update $Y_{k+1} = Y_k + x^+; k = k + 1$
4. Go to 2

$J(x)$ is the evaluation function that attempts to be maximized by the algorithm. For this reason $J(x)$ typically returns the accuracy of the classifier used on a specified training set. Assuming 10-fold cross validation, to narrow down a feature set of 4 to 2 requires the classification algorithm to run 90 times (9 different feature subsets to be tested across 10 classifications for cross-validation) as shown in Figure 1. For this reason, sequential feature selection forward works best when the optimal feature subset is small, and the classification algorithm does not have to run many times.

3.2 Sequential Feature Selection Backward

The second algorithm tested is extremely similar to the first, but aims to solve one of the issues with sequential feature selection forward. To begin, this algorithm is also a wrapper algorithm that performs a greedy state space search over all feature subset permutations. Though it has the same pitfalls as sequential feature selection forward where the algorithm is slow to run, can get stuck in local maxima, and is dependent on the specified classifier, sequential feature selection backward works best when the optimal feature subset is large instead of small. This is due to the slight change between the sequential feature selection forward algorithm and the sequential feature selection backward algorithm [3]:

1. Start with the full set $Y_0 + x$
2. Remove the worst feature $x = arg\ max_{x \in Y_k} J(Y_k - x)$
3. Update $Y_{k+1} = Y_k - x, k = k + 1$
4. Go to 2

Instead of starting with the empty set, sequential feature selection backwards starts with the full set of features. From there one feature at a time is removed and the accuracy is reevaluated. Since this algorithm effectively starts at the other end of the same greedy state space search that sequential feature selection forward uses, it will perform better when the optimal feature subset is large. Figure 2 demonstrates what this state space search looks like to narrow down a feature set of 4 to 2.

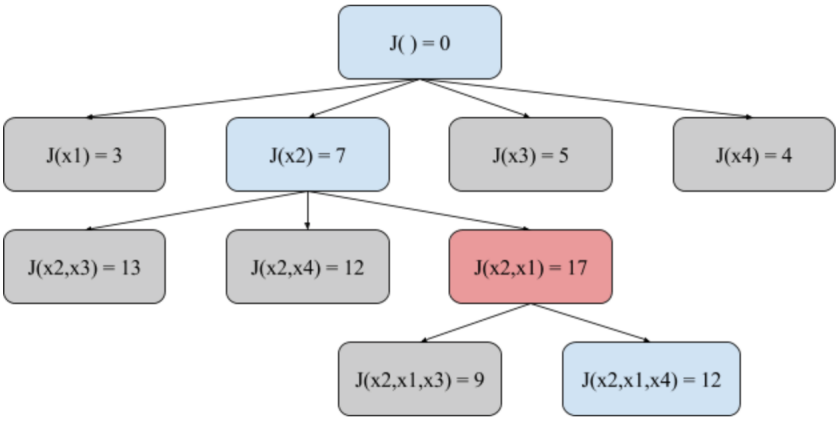

Figure 1: This shows the feature selection forward state space search for a 4-feature dataset. The blue boxes represent the $max\ J(x)$ value for each level, and the red box represents the $max\ J(x)$ value overall. The feature subset in the red box is returned as the optimal feature subset. In total, $J(x)$ must be calculated 9 times, and each calculation requires 10-fold cross validation, yielding 90 classification calculations in the above depiction.

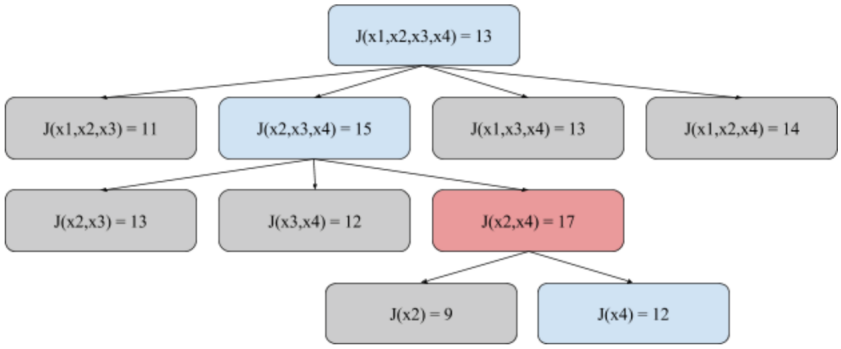

Figure 2: This shows the feature selection backward state space search for a 4-feature dataset. The blue boxes represent the $max\ J(x)$ value for each level, and the red box represents the $max\ J(x)$ value overall. The feature subset in the red box is returned as the optimal feature subset.

3.3 ReliefF

The final feature selection algorithm is different than the previous two. Instead of a wrapper algorithm dependent on classifiers, it is a pre-processing algorithm that analyzes features independent of classifiers. Because of this, reliefF selection runs extremely quickly. Additionally, reliefF selection can handle multi-class problems and incomplete data. Neither of these advantages are being implemented in this project, so the methods used to implement such functionality will be absent in the description of the algorithm. Unlike the other two feature selection algorithms, the output of reliefF is simply a ranking of all features. This takes the form of a vector where the indices correspond to the feature numbers and the value corresponds to the feature's computed weight. The highest weighted feature is ranked 1, and the lowest weighted feature is ranked last. Typically this algorithm is used to tune features by keeping all features with a weight higher than an arbitrary τ value. For this project, the top 3, 5, and 10 ranked features were chosen for analysis regardless of τ values [7].

1. Set all weights $W[A] = 0$
2. For $i = 1$ to m:
   (a) Randomly select an instance R_i
   (b) Find k nearest hits H_j
   (c) Find k nearest misses M_j
   (d) For $A = 1$ to a (iterate over every feature)
   $W[A] = W[A] - \sum_{j=1}^{k} diff(A, R_i, H_j)/(m*k) + \sum_{j=1}^{k} diff(A, R_i, M_j)/(m*k)$,

where $diff(A, I_1, I_2) = \frac{|value(A,I_1) - value(A,I_2)|}{max(A) - min(A)}$, nearset hit: a nearest neighbor with the same label as instance R_i, nearest miss: a nearest neighbor with the opposite label as instance R_i.

The above algorithm is a version of reliefF selection that only works for complete data and binary classification. The general idea of reliefF selection is to weigh features according to how well they distinguish between data points that are near each other. For example, if two data points with the same label are close by and their value for feature A differs, then feature A separates two instances of the same class, which is not desirable. Therefore, the weight of the feature would be decreased. Alternatively, if the two data points had different labels and the value of feature A also differed, then feature A would separate two instances of different classes, and its weight should be increased [7]. ReliefF selection as described above has two tunable attributes. The first is the value m in step two. It chooses how many times the weight value is updated before it is returned. In our analysis, m was set to the number of data points in the training set. The second tunable value is the k variable mentioned first in step 2b. It chooses how many neighbors are used per data point when calculating the weight vector in step 2d. In our analysis, this value was kept at 10.

3.4 Results

After running the feature selection algorithms paired with the different classifiers over the two datasets, we found that feature selection paired with KNN classification yielded the highest accuracy, better than using KNN without feature selection. From this analysis we can come to some interesting conclusions about each dataset.

For the mushroom dataset, sequential feature selection forward and sequential feature selection backward returned the same results. They both chose only 5 features which included cap color, bruises, odor, stalk surface below ring, and spore print color. As the original dataset consisted of 20 features, this is a significant improvement. Since both of these feature selection algorithms returned the same result, we can also state that sequential feature selection forward was more efficient, since it called the classification algorithms far fewer times to select a five feature subset compared to sequential feature selection backward. ReliefF selection performed equally well, however it required using the top 10 ranked features to get the same accuracy. Shockingly, the top 10 ranked by reliefF selection features did not include 3 of the 5 attributes selected by the sequential feature selection algorithms.

For the voice dataset, sequential feature selection forward and sequential feature selection backward returned different results, which is due to the fact that one (or both) fell into local maxima of the state space search. Sequential feature selection backward narrowed down the 20 attributes to 17, so it was not nearly as helpful as sequential feature selection forward, which narrowed down 20 attributes to 5. The 5 features included the first quantile in KHz, interquartile range in KHz, skew, spectral flatness, and peak frequency. ReliefF selection performed just as well, but was able to narrow down the features to 3. Two of these features were included in sequential feature selection forward (interquartile range and peak frequency), but the standard deviation of the frequency was added.

4 Classifier Performance Analysis

4.1 Setup

The classifiers described below were trained prior to using them in conjunction with sequential feature selection algorithms. A separate set of classifiers was trained on data with reliefF-reduced features. We split the data into train and test sets 80% and 20% respectively. We further separated 20% of train data into a validation set. The classifiers' specific parameters were tuned on the validation set. The

final error rate that we present was obtained by classifying the test data with the optimal classifier. The classifiers were first tuned, trained and tested on the original data with 20 features, the error rate and optimal constraints were recorded. Then we repeated the procedure on the data reduced by reliefF selection, since, as mentioned above, reliefF selection is independent from classifier. Experimentally we found out that the best results are achieved with 3 features for voice data and 10 features for mushroom data.
Since the sequential feature selection algorithms, both forward and backward, are wrapper algorithms and depend on classifier, we used the optimal classifier trained on the original data to classify the test set with reduced features.
We chose four groups of classifiers: Kernel Support Vector Machine, K-nearest Neighbor classifier, Kernel Perceptron and Naive Bayes. We now will describe performance of each group.

4.2 Kernel SVM

We used Matlab's built-in Kernel SVM, varying soft margin constraint C and kernel function. In addition, we tested different polynomial orders for polynomial kernel, and scaling factor σ for radial basis function kernel.
For linear kernel we first found the best value of C from set $\{10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}\}$, $C = 10^{n}$. Knowing that we performed fine search comparing results of classifier with C from set $10^{n}\{1, 2, 4, 8\}$ on validation set. Linear classifier didn't converge with $C >= 10$ for linear kernel for both datasets, and with $C >= 0.05$ for polynomial kernel for voice dataset.
Having two parameters for each polynomial and RBF kernels, we performed grid search for an optimal pair of parameters. In case of polynomial kernel, we considered polynomial order from set {2,3,4,5], in case of RBF kernel we used sigma from set $\{10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}\}$, and for both functions parameter C as specified above.
After finding the optimal pair for polynomial kernel $(C = 10^{n}, p = N)$, we further refined C by testing all values from set $10^{n}\{1, 2, 4, 8\}$ against order $p = N$.
Having the optimal pair for RBF kernel $(C = 10^{n}, \sigma = 10^{k})$, we did fine grid search for both C and σ from sets $10^{n}\{1, 2, 4, 8\}$ and $10^{k}\{1, 2, 4, 8\}$ respectively and obtained the best performing pair.
After obtaining the optimal parameters for each kernel function, we trained linear, polynomial and RBF kernel SVM classifiers for both datasets and used them to classify the examples from the test set.
We repeated the process on the data reduced by reliefF selection. Sequential feature selection algorithms were used with the optimal classifier of the original data, therefore we avoided the parameter specification in the table with exception of parameter C for linear kernel. A linear kernel SVM with C=1 or 2 didn't converge on the voice data with SFS-reduced features, so C was adjusted. The error rate and optimal parameters for all variations of data features are shown in Table 1 and Table 2 for both datasets.

Table 1: Error rate of kernel SVM on test *voice* data, and corresponding optimal parameter values

	Original data	ReliefF	SFSF	SFSB
Linear	0.0300	0.0253	0.0253	0.0221
	$C = 1, 2$	$C = 0.1$	$C = 0.1$	$C = 0.1$
Polynomial	0.0284	0.0221	0.0237	0.0191
	$C = 0.04, p = 2$	$C = 0.04, p = 2$		
RBF	0.0174	0.0142	0.0300	0.0221
	$C = 1, \sigma = 2$	$C = 10, \sigma = 1$		

4.3 KNN

We used Matlab's built-in K-nearest neighbor classifier. The parameters were tuned by utilizing grid search of the combination of K and methods to calculate the distance between data points. We varied K from set {1,3,5,7,9,51,101}. We used the following methods to calculate the distance between the data points: Euclidean, standardized Euclidean, cosine and Chebychev. The optimal combination for voice data is $k = 5$ and standardized Euclidean distance, other methods to calculate the distance resulted in much higher error rate - 30-50%. $k = 1$ combined with either Euclidean or cosine

Table 2: Error rate of kernel SVM on test *mushroom* data, and corresponding optimal parameter values

	Original data	ReliefF	SFSF	SFSB
Linear	0.0560	0.2044	0.0265	0.0603
	$C = 0.01$	$C = 0.01, 0.1$		
Polynomial	0.0142	0.0456	0.0683	0.0683
	any $C, p = 2$	any $C, p = 2$		
RBF	0.0948	0.0185	0.0493	0.0049
	$C = 0.8, \sigma = 2$	$C = 10, \sigma = 2$		

distances produced the smallest error rate for mushroom dataset. The error rates for both datasets are presented in Tables 3 and 4.

Table 3: Error rate of KNN on test *voice* data

	Original data	ReliefF	SFSF	SFSB
KNN, k=5	0.0302	0.00003	0.00005	0.00002

Table 4: Error rate of KNN on test *mushroom* data

	Original data	ReliefF	SFSF	SFSB
KNN, k=1	0.0139	0.00002	0.00004	0.000006

4.4 Naive Bayes

For this project we used the Naive Bayes algorithm implemented earlier in this class. The parameters of this algorithm are a prior estimate of p_i, $\widetilde{p}$, and an integer n, both used to calculate the probability of each feature in the formula

$$p_i = \frac{(number\ of\ points\ with\ x_i = 1) + n\widetilde{p}}{number\ of\ points\ + n}$$

This algorithm performed very poorly for both datasets, resulting in error rate 0.4803 for voice dataset with parameters $n = 3$, $p = 0.5$ and in error rate 0.3122 for mushroom dataset, invariant regardless of the parameters. Due to such poor performance we did not include this classification method in feature selection process and analysis.

4.5 Kernel Perceptron

We chose to use kernel perceptron implemented in class for this group of classifiers. As with kernel SVM, we tested linear, polynomial and RBF kernels. Since we implemented the simplified version of kernel perceptron, there is no parameter to tune for the linear version. The polynomial version allowed us to select the best performing order, and RBF - the optimal σ. The results are shown in Tables 5 and 6.
We attempted to run the sequential feature selection algorithms using kernel perceptron as a classifier, however this task was not completed due to exceeding time constraints. We deemed this combination of data, classifier and feature selection algorithm not practical since the results can be achieved by a different combination. It is possible that the reason for such high running time is not the complexity of the algorithm but its implementation.

4.6 Comparison of the Classifier Performance on Original Data vs Data with Reduced Features

Overall, we were able to achieve low test error for both datasets, with original number of features and with reduced features. The accuracy of the classifiers on voice data is shown in Figure 3. The

Table 5: Error rate of kernel perceptron on test *voice* data, and corresponding optimal parameter values

	Original data	ReliefF	SFSF	SFSB
Linear	0.3096	0.0806		
Polynomial	0.4850	0.0284		
	$p = 2$	$p = 4$		
RBF	0.1769	0.0300		
	$\sigma = 10$	$\sigma = 1$		

Table 6: Error rate of kernel perceptron on test *mushroom* data, and corresponding optimal parameter values

	Original data	ReliefF	SFSF	SFSB
Linear	0.0499	0.0696		
Polynomial	0.0203	0.0363		
	$p = 2$	$p = 5$		
RBF	0.0123	0.0314		
	$\sigma = 10$	$\sigma = 1$		

error rate for all SVM classifiers ranged in 1.4-3%. SVM classifiers performed very similarly on the original data and reliefF-reduced data with gradual increase in performance from linear to RBF, with smaller error rate on reliefF-reduced data across all kernels. Although polynomial kernel's performance was average comparing to linear and RBF kernels for original reliefF-reduced data, it produced the smallest error rate on SFS-reduced data. The lowest error rate on the original data is 1.74% and the lowest error rate overall (by kernel SVM) is 1.42%, the latter achieved by RBF SVM on reliefF-reduced data with only 3 features.

The accuracy of the classifiers on mushroom data is shown in Figure 4. There is much more variance, the accuracy ranges from 0.49% to 9.48%. There are no clear tendencies in the classifier performance, the lowest error rate is produced by a different classifier for any given variant of the dataset. Linear and RBF kernels performed better on the feature-reduced data regardless of the feature selection method. The lowest error rate among SVM classifiers on original data was achieved by polynomial kernel order 2; the overall lowest rate was achieved by RBF kernel on SFS-Backward-reduced data - 0.49%.

KNN classifier on feature reduced data exponentially decreased error rate for both datasets, making KNN on the data with selected features the most efficient method of classification for the given data. Kernel perceptron proved very effective on reliefF-reduced voice data, in comparison with the same method used on original data (Figure 5). As far as performance of mushroom data, it is slightly better on the original data, although error rate on the data with reduced features stayed within reasonable margin (Figure 6).

5 Conclusions

In our experiment we were able to build classifiers for both datasets with very high accuracy, using the data with the original number of features. The erroneously classified data accounted for about 1.5% on each test subset. After experimenting with three different techniques of feature selection, we found that all three of them, although produced slightly varying intermediate results - selected features, were very effective. By reducing the number of features, we were able to reduce the error rate to a fraction of a percent.

Working with two binary datasets with the same original number of features, we noticed that there is no correlation between classifier and its performance on those datasets. We concluded that classifiers do not necessarily perform similarly on similarly structured data. In order to obtain the most accurate and fast classifier, it is imperative to explore broad spectrum of options among classifiers as well as dimensionality reduction techniques.

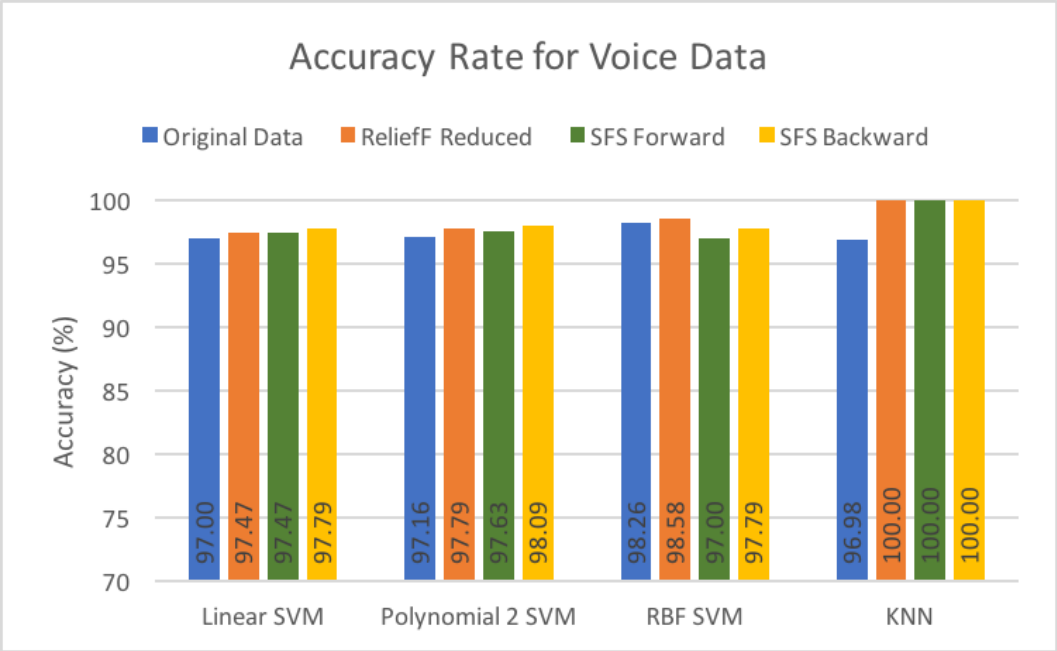

Figure 3: Accuracy rate for *voice* data

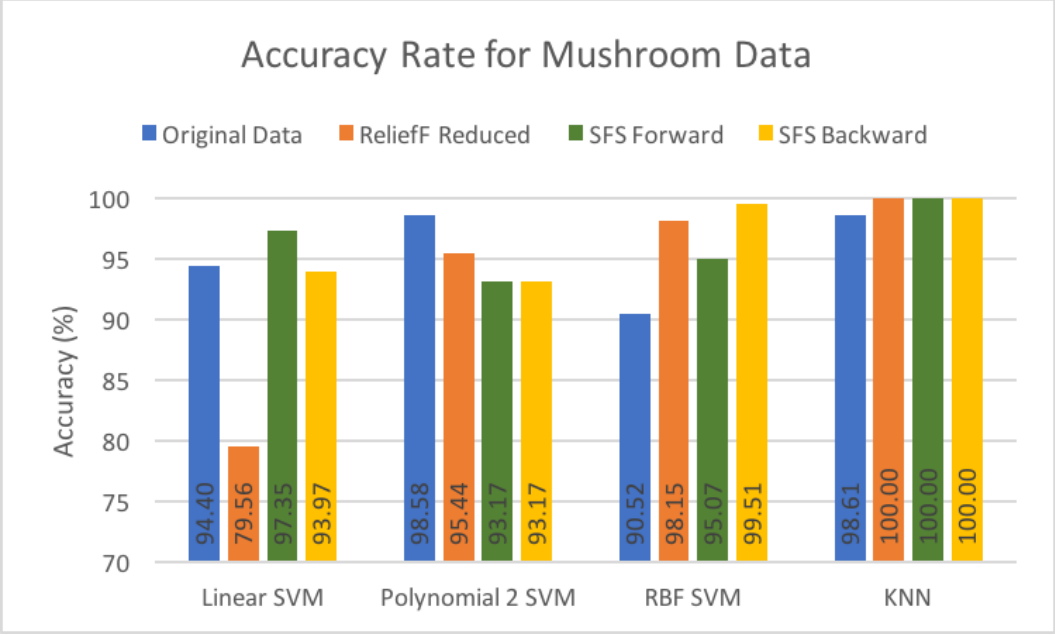

Figure 4: Accuracy rate for *mushroom* data

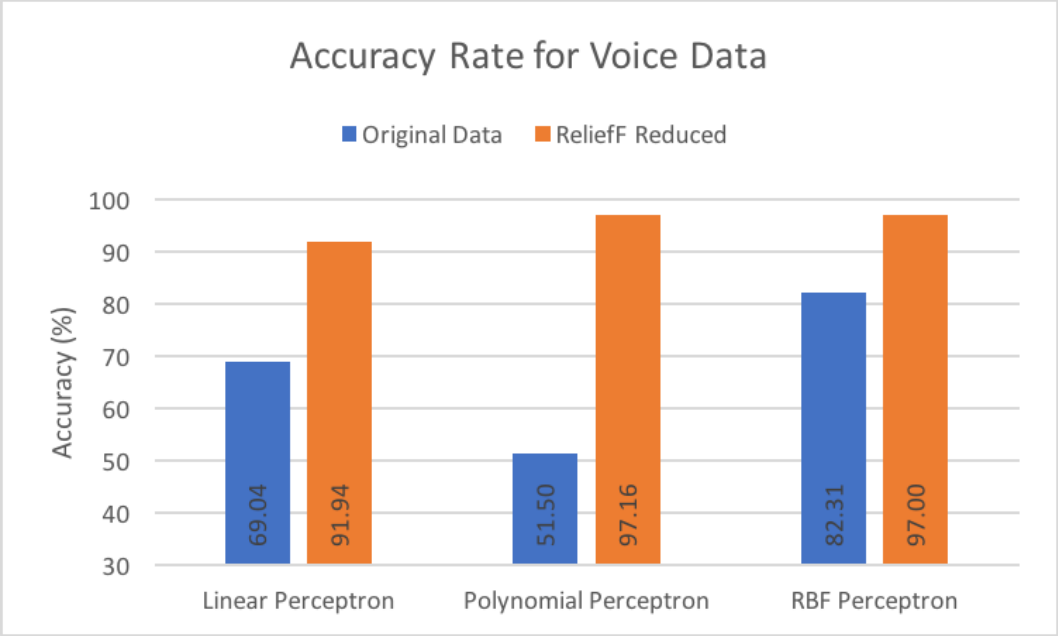

Figure 5: Accuracy rate for *voice* data using Kernel Perceptron

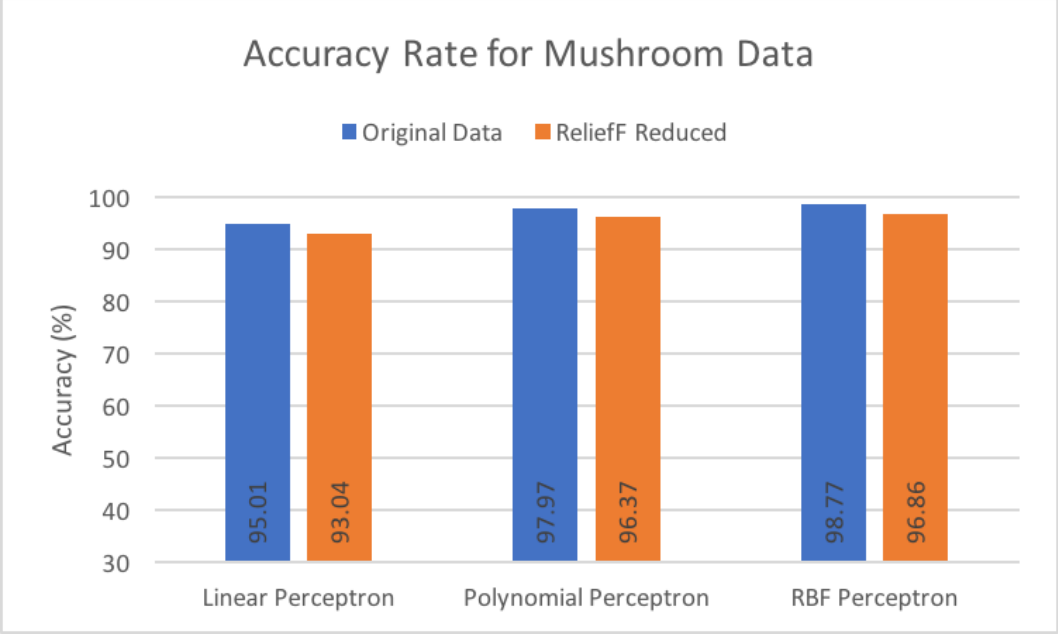

Figure 6: Accuracy rate for *mushroom* data using Kernel Perceptron

References

[1] Becker, K. Gender Recognition by Voice [https://www.kaggle.com/primaryobjects/voicegender].

[2] Durgabai, R. Feature Selection using ReliefF Algorithm. *International Journal of Advanced Research in Computer and Communication Engineering.* Vol. 3, Issue 10, October 2014

[3] Gutierrez-Osuna, R. Sequential Feature Selection. *CSCE 666 Pattern Analysis.* Computer Science & Engineering, Texas A & M University.

[4] Kononenko, I., Simec, E., Robnik-Sikonja, M. Overcoming the Myopia of Inductive Learning Algorithms with RELIEFF. Applied Intelligence (1997) 7: 39. doi:10.1023/A:1008280620621

[5] Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[6] MathWorks - MATLAB and Simulink for Technical Computing. [https://www.mathworks.com]

[7] Robnik-Sikonja, M., Kononenko, I. Theoretical and Empirical Analysis of ReliefF and ReliefF. *Machine Learning Journal* (2003) 53:23-69

[8] Yu, L., Liu, H. Feature Selection for High-Dimensional Data: A Fast Correlation-Based Filter Solution. Department of Computer Science & Engineering, Arizona State University.