SVM and Random Forest Implementation Using Scikit-Learn

## **Implementation**

Using scikit-learn, which developed from the libsvm source code. Libsvm source code uses the ovo scheme and consists of four kinds of kernel functions, including the Linear function, polynomial function, radial basis function and sigmoid function. The most popular one is rbf, which always performs the best according to the experience. But the linear kernel performs better in our project so we choose the linear one.

In random forest, we also developed it from scikit-learn, with a random forest classifier source code which is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting.

## **Advantages and Disadvantages**

**SVM**

It is said that the most important principle of SVM is to find the best separating hyperplane, which will maximize the margin of support vectors. Its maximum of margin is different according to its choice of different loss function.

It is also constructed on the foundation of principle of SRM, which is so called Structural Risk Minimization. So SVM possesses a bunch of advantages that keep it the most prevalent algorithm when coming to high VC dimension problems. It can avoid over-fitting and is better at generalizing.

Meanwhile, as it is most sensitive to the kernel function and barely using probability measurement or Law of Large Numbers ext., once it has been trained, it can perform well. Which leads to the result of nice performance implementing with small samples.

It is also run fast and easy to learn. But there are also multiple disadvantages about it.

First it can be told of as a side effect of its advantage, in that SVM can’t implement well large training sets. Due to the fact that SVM solve support vector with quadratic function, and it need the calculation of n order matrix, where n is the number of samples. So it requires more time for the large training set.

Second, it requires ova or ovo decision scheme as implementing multi-classes problems so it can’t stay as original binary classification.

Third, as its probability mostly decided by its kernel function, it is sensitive to losing data (for example when implementing semi-supervised problem) and its choice of parameters and kernel functions. Its performance can be randomly incurred with practical problems.

**Random Forest**

The main idea to process Random Forest is to generate numerous random decision trees and get the mode from each tree. In common, we can do it in two ways: bootstrap sampling or randomly picking all features as a subset in every node in order to calculate the most suitable way to divide it.

For advantages, with characters that it doesn’t need to do feature selection (due to it’s select features subset randomly), Random Forest works efficiently on high dimension data (many features). And after trained, it can show that which features are most effective as it provides two ways to choose features: mean decrease impurity and mean decrease accuracy; both will show the features which are less effective.

And as it constructs trees, the generalization error is based on Unbiased Estimation and makes it stronger in model generalizing as well.

Also, due to it’s advantage in training speed, Random Forest is suitable for parallel computing as different trees are separated at the same time.

After all, it still works well as we can get good accuracy even we lose some of the data and also can balance error in an unbalanced data set.

On the other hand, when using Random Forest processing regression problems with larger noise, often over-fitting will occur. In other words, it only has a good estimation in training set which cannot be seen on the other data set. Although we can fix these problem by Increasing the data set amount, decreasing the number of features, or regulating features, it is not the first choice for these data sets.

Another disadvantage of Random Forest is that if different features exist, different values divide and the features with more division will inevitably be more influential on the result. That is, Random Forest is not as reliable for these types of data sets.

## **SVM vs. RF**

In practical terms both have their advantages and disadvantages. So generally speaking, we couldn't say which one is better than the other, it depends on data and its distribution. For a two class problem and when expecting the data to be reasonably clean and outlier free, structural risk minimization is a powerful approach and SVM may perform better.

Whereas when coming to multi-classes case and especially expecting to have outliers, then Random Forest should be better choice. As similar, RF performs better than SVM when implementing with large data set and it can be more easy to extend as well run in faster processing speed. But when dealing with a limited small data set, SVM should be better. SVM also been reported to be the best algorithm working for text or email classification.

**SVM Background**

The first SVM algorithm, also known as the original maximum-margin hyperplane algorithim, was developed in 1963 by Vladmir N. Vapnik and Alexey Ya. Chervonekis. Automatic learning from examples has been a long-standing goal in the field of computer science. However with the development of the World Wide Web, the amount of information that is available and needs processing has exploded. There was an increasing need to extract and analyze the structures from the data. SVM finds the most optimal hyperplane to categorize new examples based off labeled training data. SVM has developed is capable of yielding efficient classification results from complex and noisy data. It separates classes by maximizing the margins between the classes. This used to be only a linear separation known as a “linear classifier.” SVM, given a set of training examples (which are marked for one category or another), builds a model which will assign new examples into one of the categories. Finally in 1992, Vapnik helped develop a method to create nonlinear classifiers using the kernel trick. Nonlinear classifiers now allow for the best possible predictive performance and much more accurate results.

In essence, a SVM is a discriminative classifier defined by a separating hyperplane. The operation of the SVM is to find the hyperplane which gives the largest minimum distance to the provided training examples.This creates an efficient method to classify and predict new data.

The notation used to formally define a hyperplane is f(x) = 𝛃0 + 𝛃Tx where 𝛃 is called the weight vector, 𝛃0 is called the bias and x represents the training examples closest to the hyperplane. The first representation chosen for the hyperplane is defined as |𝛃0 + 𝛃Tx| = 1. Further calculations are done to determine the distance between a point x and the hyperplane which results in the formula: M = 2/||𝛃|| where M represents the margin that must be maximized to find the maximum-margin hyperplane. Maximizing M is the same as minimizing L(𝛃) which finalizes the formula as minL(𝛃) = 0.5||𝛃||2 subject to yi(𝛃0 + 𝛃Txi) >= 1 where yi represents the labels of the training example.

**Random Forest Background**

Random Forest is the set of many classification trees. To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

Each tree is grown as follows:

1. If the number of cases in the training set is N, sample N cases at random - but *with replacement*, from the original data. This sample will be the training set for growing the tree.

2. If there are M input variables, a number m<<M is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.

3. Each tree is grown to the largest extent possible. There is no pruning.

Reducing m reduces both the correlation and the strength. Increasing it increases both. Somewhere in between is an "optimal" range of m - usually quite wide. Using the oob error rate (see below) a value of m in the range can quickly be found. This is the only adjustable parameter to which random forests is somewhat sensitive.

**Random Forests Parameters**

In random forests, forest classifiers have to be fitted with two arrays: a sparse or dense array X of size [n\_samples, n\_features] holding the training samples, and an array Y of size [n\_samples] holding the target values (class labels) for the training samples.

There are primarily 2 features which can be tuned to improve the predictive power of the model :

**(A) n\_estimators** :

This is the number of trees in the forest, which means it build before taking the maximum voting or averages of predictions. Though higher number of trees could give better performance,it makes the code slower. Usually, we choose as high value as our processor can handle so that can makes the predictions more stable.

**(B) max\_features**:

This is the size of the random subsets of features to consider when splitting a node. The lower the greater the reduction of variance, while the greater the increase in bias. There are multiple options available in Python to assign maximum features. Here are a few of them :

**Auto/None** : This will simply take all the features which make sense in every tree.

**sqrt** : This option will take square root of the total number of features in individual run. For instance, if the total number of variables are 100, we can only take 10 of them in individual tree. “log2” is another similar type of option for max\_features.

**0.2** : This option allows the random forest to take 20% of variables in individual run. We can assign and value in a format “0.x” where we want x% of features to be considered.

On most cases, we use max\_features=n\_features for regression problems while max\_features=sqrt(n\_features) for classification tasks. Usually,when setting max\_depth=None in combination with min\_samples\_split=1 (i.e., when fully developing the trees) could achieved good results. However, these values are usually not optimal, and might result in models that consume a lot of ram. The best parameter values should always be cross-validated.

**Results**

SVM

C: 1.0

Kernel: Linear

Gamma: 0.0001

Decision function shape: ovo

Degree: 3

SVM%20-%20Chart.pdf

Random Forest

Estimators: 10,

Criterion: Gini

Max depth: None

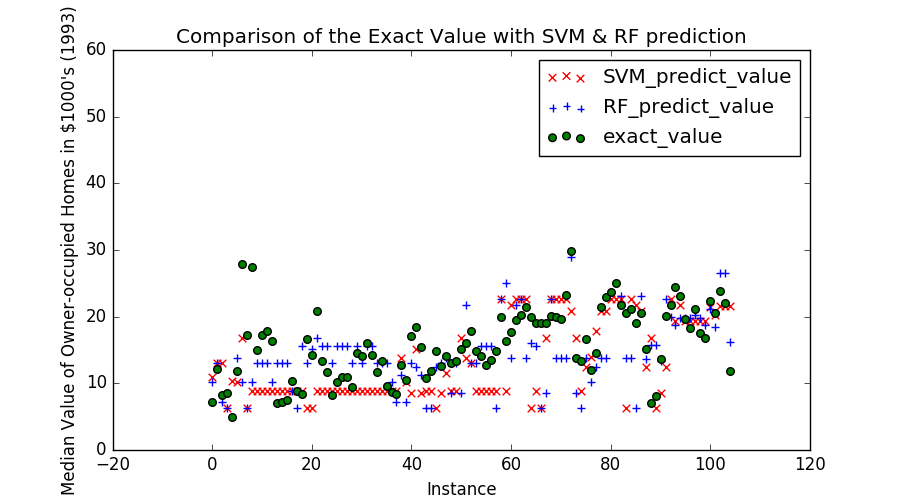
Min samples split: 2

Min samples leaf: 1

Min weight fraction leaf: 0.0

RF%20-%20Chart.pdf

Combination Plot



End Result

Given the current data set of the median value of owner-occupied homes in 1000’s of dollars as of 1993, both methods, random forest and SVM, were able to be successfully used. However, one of the key problems with random forest was observed where it attempts to over-fit. On the other hand, SVM varied less and in doing so was better able to predict the actual value. The training data set was rather large (400) when compared to the test set (106) though this seemed to be close to ideal for being able to make accurate predications. For the current settings listed above, the mean absolute error for SVM was 4.14 and for random forest was 4.52.