# CSE 517A Machine Learning

# Application Project Report

Chunyuan Li
Jiarui Xing

## 1. Introduction

The aim of this project is to predict the forest cover type (the predominant kind of tree cover) from strictly cartographic variables (as opposed to remotely sensed data). The dataset come form UCI machine learning repository [1], which contains both cover types and features. Physically, each sample describes the environmental properties of a 30 x 30 meter cell in actual forest.

**Dataset statistics.** The dataset contains seven cover types and 54 features. Among the features, ten of them are continuous value and the rest 44 features are binary values. The dataset has 15120 labeled samples and the number of samples in each class (cover types) are equal, which ensure a balanced dataset. We analyzed correlation between every two features. The high correlated features can be further processed to decrease the number of dimensions. Thus, decrease the complexity of the model. For each feature with continuous value, we calculated different statistical measures, including mean, standard deviation, 25% percentile, 72% percentile etc. We also used skewness to see how asymmetric the distribution of each feature can be. Due to limitation of the space, we will not show all the data here. The complete result of statistical analysis can be found in our GitHub repository1.

**Implementation efforts**. We tried supervised learning, clustering, semi-supervised learning and dimension reduction on our dataset. In the supervised learning section, we trained 7 classifiers and evaluated them based on accuracies, statistical tests and efficiencies. In the clustering section, we explored parameter selection and evaluated the performance of clustering on our dataset. In the semi-supervised learning section, we tested the usefulness of label propagation on partly labeled dataset. In the dimension reduction section, we tried PCA, SVD and autoencoder with different target dimensions and used the low-dimensional data on supervised learning. We used scikit-learn, pandas, nltk and several other Python tools to complete this project.

In the rest of the report, we will introduce the discuss and compare all the classification methods by group. Section 2 will compare and analyze all the supervised method we used. Section 3 is unsupervised method. Section 4 will show the implementation of semi-supervised method. The last section will conclude all the classification method we used and talk about future work.

## 2. Supervised Learning

**Used models.** We tried the following classifiers:

- Ridge classifier (linear classifier with L2 regularization)
- Lasso classifier (linear classifier with L1 regularization)
- Gaussian process
- SVM with linear and RBF kernel

1 File ‘dataset statistical discription.txt’ in
https://github.com/Remussn/CSE517˙Project/tree/master/milestone˙1

- Logistic regression
- CART decision tree (with and without bagging)
- Neural network (with 2 layers)

**Hyperparameter search.** We used telescope search and cross validation to select the hyperparameters of SVM kernel. For Gaussian process kernels, however, due to the huge time cost for training Gaussian process, we used RBF kernel with default hyperparameters.

**Performance evaluation.** We record the 10-rerun average of 10-fold cross validation accuracy, test accuracy, training and test time. We also did a sign test based on 10-rerun 10-fold cross validation result. Since Gaussian process uses a lot of memory, we have to train all the classifiers using a small batch of 1000 samples from full dataset. The results are:

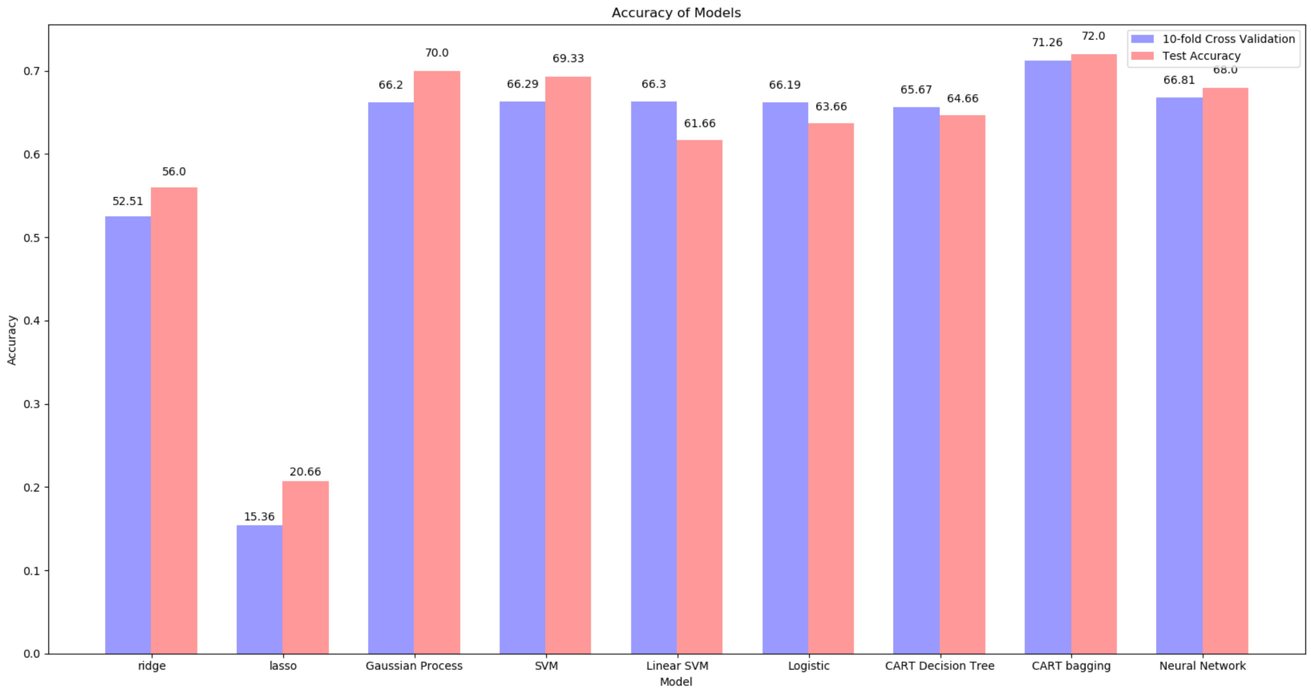

*Figure 4.1 Accuracies of different models*

As we can see from figure 4.1, the accuracies of nonlinear models are quite close, among which CART decision tree with bagging has both highest cross validation and test accuracy. Linear SVM preforms best among the linear models and has quite high cross-validation accuracy.

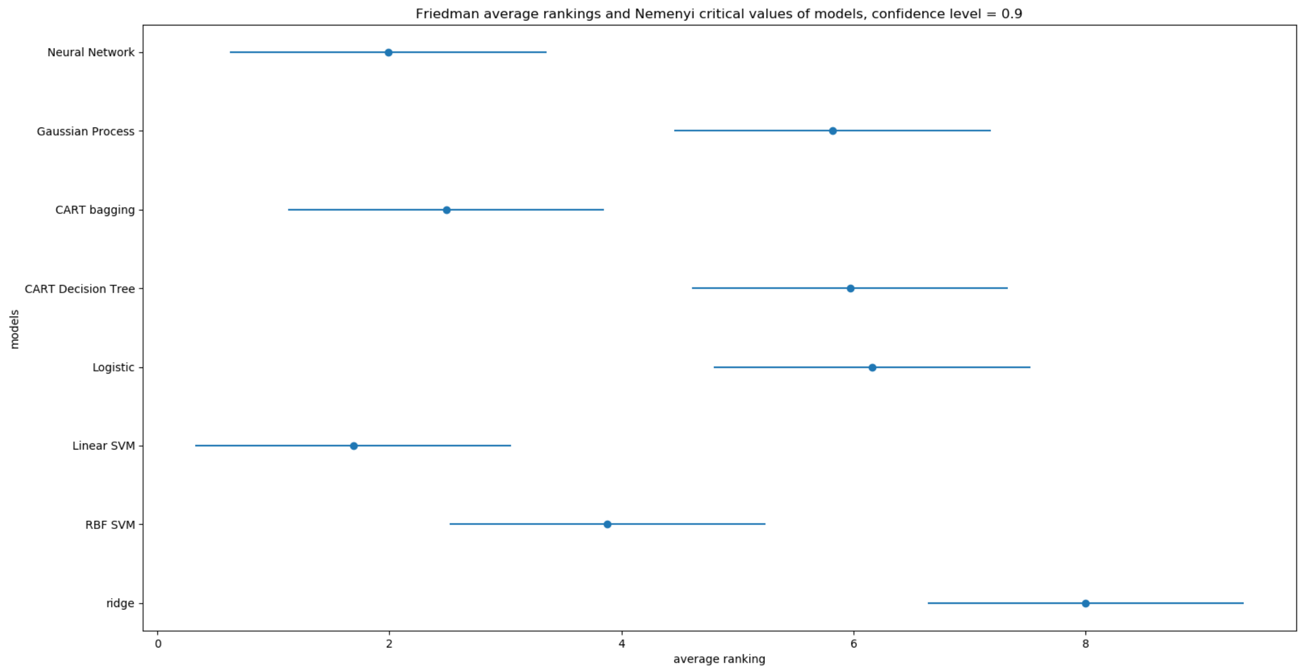

*Figure 4.2 Sign test result of different models*

**Statistical test.** We used Friedman test and Nemenyi post-hoc test as our statistical test. We trained our models on 10 re-run 10-fold cross validation and computed their average ranks. The result is shown in figure 4.2, where the points represent the average ranks. The width of error bars beside the average ranks are the critical values of Nemenyi test and having overlap area between bars of two classifiers means that we do not have confidence - over confidence level - to say they have statistical difference in average rank. As is shown is figure 4.2, when confidence level is 90%, linear SVM, neural network (with same architecture as in milestone3) and CART decision tree with bagging perform statistically better than Gaussian process, CART decision tree, Logistic regression and ridge regression. Though (quite surprisingly) linear SVM has the highest average rank, statistically it has no statistical difference with the following 3 classifiers (with 90% confidence).

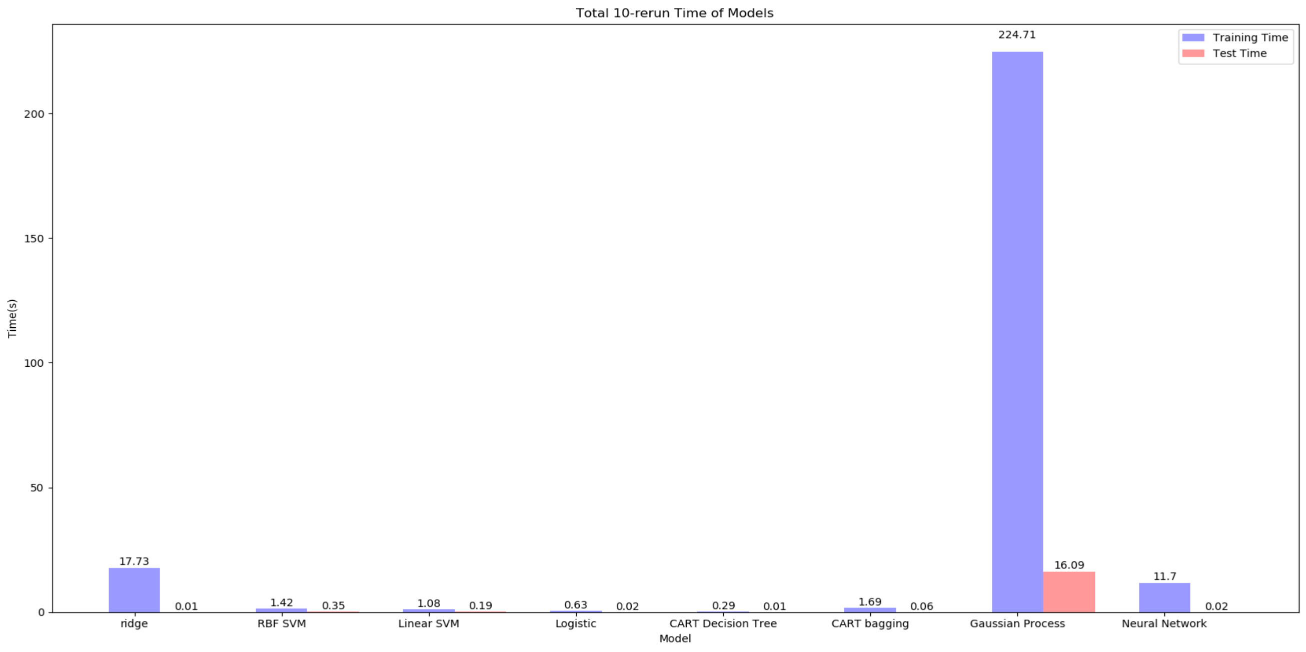

*Figure 4.3 Training and test time of different models*

**Efficiency evaluation.** Figure 4.3 shows the total training and test time of 10-rerun of different models. We can see Gaussian process is slow (for both training and test time). The fastest method is CART decision tree. Considering it gives us pretty good accuracy, it is the best model if we want to save time. CART decision tree with bagging is also quite fast and is overall the model that suits our task best.

## 3. Clustering

In this task, purity is used to evaluate the performance. It is defined as

$$purity = \frac{1}{N}\sum_{i=1}^{k} \max_{j} |c_i \cap t_j|,$$

where $N$ is the number of samples, $k$ is the number of clusters, $c_i$ is the i$^{\text{th}}$ cluster and $t_j$ represents classification type which has max count for cluster $i$. We first explored relationship between purity and the number of clusters. As it is shown in figure 5.1 below, the 'kink' point is not clear for our dataset. We use 10 clusters which is slightly larger than the total number of classes.

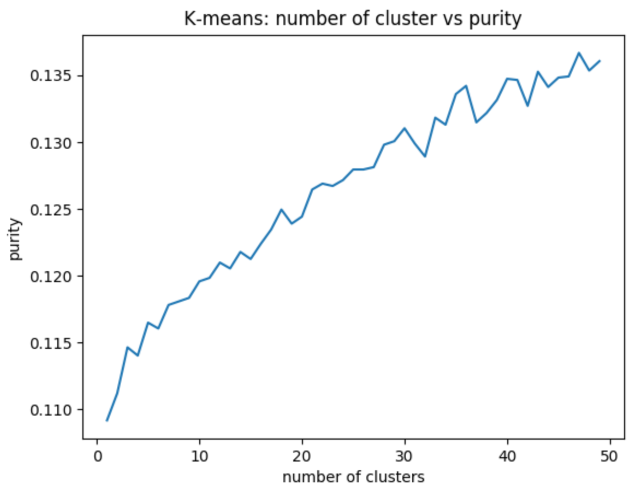

*Figure 5.1 Number of clusters vs purity*

The figure 5.2 shows the classification result of K-means and Kernel K-means where Kernel K-means has purity of 0.1194 and K-means 0.1191. The purity score for both method is very low. The low purity might due to K-means clustering assumes the variance of the distribution of each attribute (cluster) is spherical. However in our dataset, this assumption does not hold. Besides, K-means is trying to minimize the squared Euclidean distance between samples and cluster centers. Squared Euclidean distance is not a good metric in high dimensions and it will result in curse of dimension. Therefore, clustering does not suit our dataset.

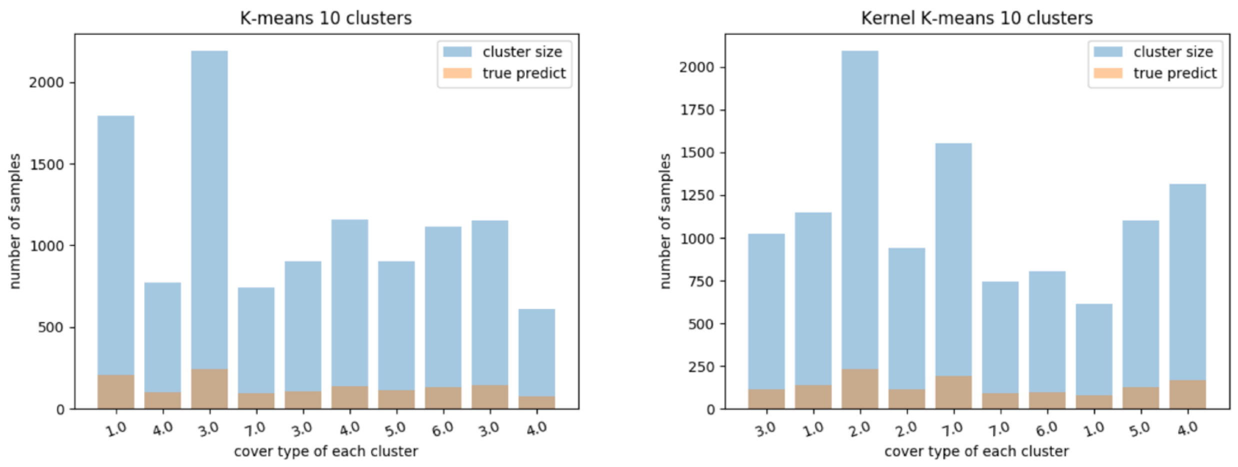

*Figure 5.2 Performance of K-means vs Kernel K-means*

## 4. Semi-supervised Learning

In this part, we first generated a partly labelled data by erasing part of labels of our raw data. Then we trained an SVM based only on the labelled data and predicted the labels of unlabeled samples by label propagation and train another SVM model to see does using the (originally) unlabeled data helps improving model performance. In details, we used the following training data:

- 10%/30%/50%/70% of raw data, all labelled
- All samples, 10%/30%/50%/70% labelled
- Full raw data (as comparison)

The result is:

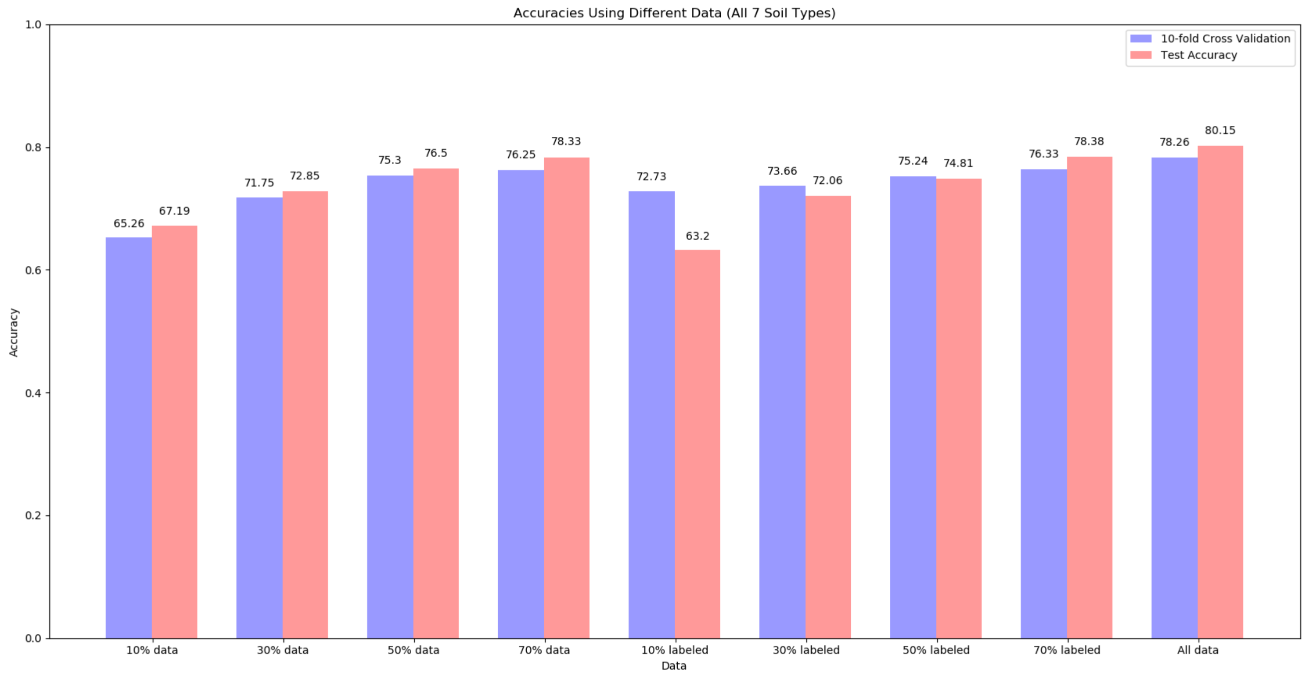

*Figure 6.1 Accuracies of different models*

As is shown in figure 6.1, the accuracies using partly labeled data are similar to that of batch data with same number of labeled samples, which indicates that the unlabeled data doesn't give help. What's more, the accuracy of 10% labeled (and 90% unlabeled) data is much worse than that of 10%

batch data, which suggests that label propagation mispredicted many labels and lowered the performance.

## 5. Dimension Reduction

**Methods**. In this section, we experimented on 3 dimension reduction methods (PCA, SVD and autoencoder). Since PCA and SVD can only work with continuous features, we used only 10 continuous features of our data for all 3 methods.

**Tasks**. We tried the 3 methods with target dimension from 1 to 9 and compute the classification accuracies (10-fold cross validation and test accuracy) using SVM under each situation. In addition, we also compute the accuracy with all 10 features as comparison. We also visualized the data when target dimension is 3.

**Autoencoder Architecture**. The autoencoder has 2 architectures under different target dimension:

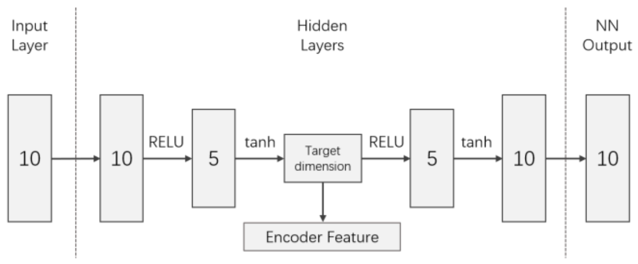

*Figure 7.1 Autoencoder Architecture (target dimension $\leq$ 5)*

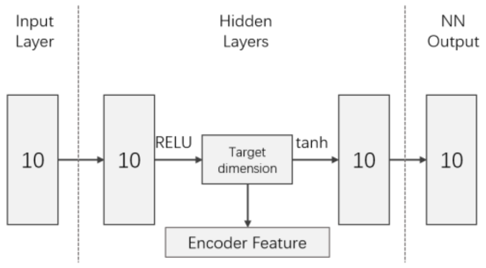

*Figure 7.2 Autoencoder Architecture (target dimension > 5)*

**Results**. The result is:

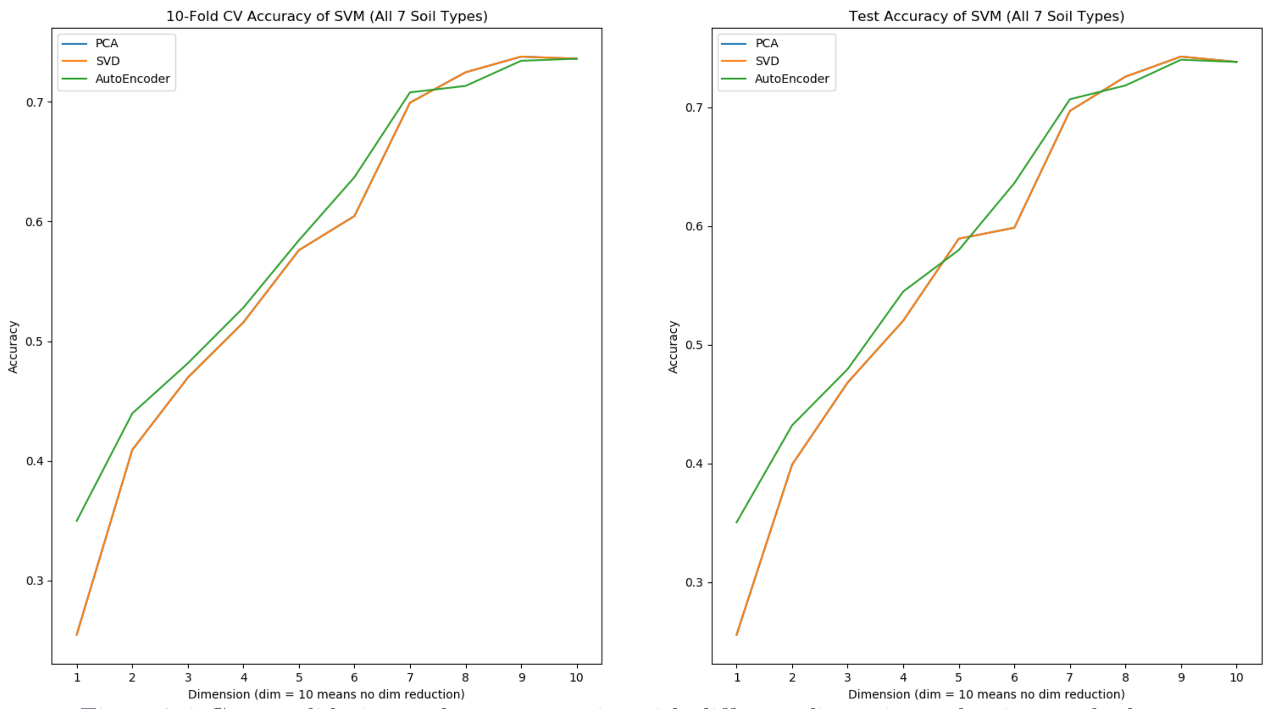

*Figure 7.3 Cross validation and test accuracies with different dimension reduction methods*

As is shown in figure 7.3, PCA and SVD gives us very close results. And an interesting observation is that autoencoder preforms much better than PCA and SVD when target dimension is low (1 and 2), but with higher targets their performance is quite close.

## 6. Summary

**Conclusion.** We did supervised learning, clustering, semi-supervised learning and dimension reduction on our dataset. In the supervised learning section, we trained 7 classifiers and evaluated them based on accuracies, statistical tests and efficiencies. Among all the classifiers, decision tree with bagging has the highest test accuracy, decision tree has the highest efficiency and linear SVM performs best in the statistical test. In the clustering section, we found that due to high dimension of feature space, clustering method cannot perform well on our dataset. In the semi-supervised learning section, we tested the usefulness of label propagation on partly labeled dataset. The result shows that in most cases, label propagation doesn't help. And when the ratio of labeled samples is small, label propagation may mispredict and lower the accuracy. In the dimension reduction section, we tried PCA, SVD and autoencoder with different target dimensions and used the low-dimensional data on supervised learning. PCA and SVD always give us similar performance. Autoencoder performs better, especially in the case that target dimension is 1. However, autoencoder use much more time than PCA and SVD.

**Lessons learned.** (1) Some machine learning method tend to consume plenty of time when training the model. Therefore, when trying to find the hyperparameter to maximize the model score, it is not wisdom to try every possible hyperparameters in brute force search. On the contrary, understanding the meaning of each parameter, as well as using non-brute-force searching strategies, will help to save our time during the experiment. (2) The novel or cutting-edge methods are not necessarily the most appropriate method for a dataset. In our experiment, some simple machine learning methods perform better than complex methods. In a word, the best algorithm is the algorithm that works.

**Future work.** Inspired by second part of learned lessons, we plan to explore on more effective hyperparameter searching methods, e.g. active search which able to automatically find the most appropriate method of the problem.

## 7. References

[1] C. Blake, E. Keogh, and C.J. Merz. UCI repository of machine learning databases, 1999. (URL: h ttps://archive.ics.uci.edu/ml/index.php).