ECE 285 Class Project Report

Based on

Source localization in an ocean waveguide using supervised machine learning

Yiwen Gong (yig122@eng.ucsd.edu),
Yu Chai(yuc385@eng.ucsd.edu),
Yifeng Bu(ybu@eng.ucsd.edu),
Xinxin Chen(xic045@eng.ucsd.edu)

Abstract

Source localization based on ocean acoustics sensor's data is critical in oceanography. In this report, we apply the paper "Source Localization in an ocrea waveguide using supervised machine learning" by Haiqing Niu, Emma Reeves and Peter Gerstoft and optimize the machine learning method for a better classification result. The main optimization strategy we use is PCA dimension reduction to extract the features of the data. And then apply Feed-forward Neural Networks (FNN) , random forests (RF) and Extremely randomized trees in this classify different distances in this range estimation problem.

1. Introduction

We first preprocess the data. This part application is provided by the lecturer. According to our test and analyzation, we found that if we directly apply feed-forward neural networks (FNN), random forests (RF) and extra trees, as the source distance increases, we get terrible results. This is due to poor sensing data collection and noise interference.

Hence, we make a PCA dimension reduction before applying these two machine learning methods for decreasing the complexity of the data so that we can estimate distance based on the most significant features.

2. Method

2.1 Input data preprocessing

Here we use the data preprocessing proposed by Haiqiang et.al[1]. The data received from sensor is an array of pressure, and needed to be transformed into a normalized sample covariance matrix. The matrix includes both amplitude and phase information instead of kinds of complex variables such as amplitude of the pressure field, transmission loss etc.

By taking the Discrete Fourier Transform of the input pressure data at L sensors, complex pressure at frequency f can be calculated. The sound pressure is modeled as

$$\mathbf{p}(f) = S(f)\mathbf{g}(f, \mathbf{r}) + \epsilon,$$

Where ε denotes noise, S(*f*) denotes source term and g is the Green's function.

The complex pressure is also normalized to reduce the effect of the source amplitude|S(*f*)| by applying

$$\tilde{\mathbf{p}}(f) = \frac{\mathbf{p}(f)}{\sqrt{\sum_{l=1}^{L} |p_l(f)|^2}} = \frac{\mathbf{p}(f)}{\|\mathbf{p}(f)\|_2}.$$

Besides, the normalized sample covariance matrices are averaged over Ns snapshots to form the conjugate symmetric matrix.

$$\mathbf{C}(f) = \frac{1}{N_s} \sum_{s=1}^{N_s} \tilde{\mathbf{p}}_s(f) \tilde{\mathbf{p}}_s^H(f),$$

Here H represents conjugate transpose operator and $\tilde{\mathbf{p}}_s$ is the sound pressure over the sth snapshot.

The data preprocessing is provided by Professor in matlab scripts.

2.2 Source range mapping[1]

Since the source localization is a classification problem, we need to discretize the source ranges into different classes or as known as labels. The label represents the actual source range class and will be the target output for the model. SVM (Support Vector Machine) and RF(Random Forest) methods could directly use the labels to train the model. However, for the FNN (feed-forward neural networks), we need to map the K*1 labels into as K *K binary vectors. The mapping method also called one-hot encoder. One hot encoding transforms categorical features to a format that works better with classification and regression algorithms.

2.3 PCA dimension reduction

Too many features in the data will cause misclassification due to complex features. Such data with a high dimension are often transformed into lower dimensional by utilizing the principal component analysis (PCA)[4]. It finds the low dimensional approximation of the data by projecting it onto linear subspaces. This kind of technique has a broad usage such as information retrieval, image processing, and genomic analysis[5]. The goal of PCA is to reduce the dimensionality of the data while keep as much as possible of the variation of the original data.

We first compute the eigenvalues of the sample covariance matrix

$C = \frac{1}{M}\sum_{n=1}^{M}\Phi_n\Phi_n^T$, where $\Phi_i = x_i - \bar{x}$

Where the original data has M samples and n features, eigenvalues are $\lambda_1 > \lambda_2 > \cdots > \lambda_N$. Their corresponding eigenvectors will also be computed as $u_1, u_2, \ldots, u_N$. These eigenvectors form a basis for $x_i - \bar{x}$. To project data onto a lower dimension K, we will only keeps terms corresponds to the K largest eigenvalues.[6]

$$\hat{x} - \bar{x} = \sum_{i=1}^{K} b_i u_i$$

The input data after preprocessing contains 7200 features, thus, we need to extract the most important features so that we can classify the data as accurate as possible. The problem is that how to choose the dimension of the PCA so that it could keep the most important features and abandon useless characteristics. We compare the PCA dimensions of from 10 to 100.

2.4 FNN

FNN works in a single forward direction. In FNN, the data moves from the input nodes, through hidden layers and to the output nodes and there are no cycles or loops in the network[2].

We directly apply python Scikit-learn module to implement FNN method. Following is an example code of implementation.

```
>>> from sklearn.neural_network import MLPClassifier
>>> X = [[0., 0.], [1., 1.]]
>>> y = [0, 1]
>>> clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
...                     hidden_layer_sizes=(5, 2), random_state=1)
...
>>> clf.fit(X, y)
MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto',
       beta_1=0.9, beta_2=0.999, early_stopping=False,
       epsilon=1e-08, hidden_layer_sizes=(5, 2), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=1, shuffle=True,
       solver='lbfgs', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False)
```

We preprocess the data and label the classes as we discussed in the preceding sections.

2.5 Random Forest

Random forest works as a large collection of decorrelated decision trees.The algorithm will first create random subsets of original training set, each will form a decision tree. Then using all decision trees to make a ranking of classifiers. At the end, a voting mechanism will be used to determine the final class of one new sample.[3]

Like FNN method, we directly apply python Scikit-learn module to implement the Random Forest algorithm.

```
from sklearn.ensemble import RandomForestClassifier
clf1 = RandomForestClassifier(n_estimators=500,random_state= 0,max_features="auto")
```

2.6 Extremely randomized trees

Extremely randomized trees, known as extra tree, is another method of the decision tree model, which segments the input data randomly into a number of regions. Then using a random

estimators to build decision tree on each sub-sets of the input data and averaging over the decisions trees to improve accuracy and control overfitting.

For each input sample Xn, n = 1, ...N. The input data will be divided into two groups using the randomly cutoff value c based on whether Xi is larger or less compared than c value. Do this process repeated.

Basic algorithm:[7]

Table 1 Extra-Trees splitting algorithm (for numerical attributes)

Split_a_node(S)
Input: the local learning subset S corresponding to the node we want to split
Output: a split $[a < a_c]$ or nothing
– If **Stop_split**(S) is TRUE then return nothing.
– Otherwise select K attributes $\{a_1, \ldots, a_K\}$ among all non constant (in S) candidate attributes;
– Draw K splits $\{s_1, \ldots, s_K\}$, where $s_i = \textbf{Pick\_a\_random\_split}(S, a_i), \forall i = 1, \ldots, K$;
– Return a split s_* such that $\text{Score}(s_*, S) = \max_{i=1,\ldots,K} \text{Score}(s_i, S)$.

Pick_a_random_split(S,a)
Inputs: a subset S and an attribute a
Output: a split
– Let $a^S_{\max}$ and $a^S_{\min}$ denote the maximal and minimal value of a in S;
– Draw a random cut-point a_c uniformly in $[a^S_{\min}, a^S_{\max}]$;
– Return the split $[a < a_c]$.

Stop_split(S)
Input: a subset S
Output: a boolean
– If $|S| < n_{\min}$, then return TRUE;
– If all attributes are constant in S, then return TRUE;
– If the output is constant in S, then return TRUE;
– Otherwise, return FALSE.

As we could see from the process above, it's almost the same process as the Random Forest except that the random forest choose the best cutting threshold for features but Extra tree choose the random cut. In real world, some data sets will work better in Extra compared to Random Forest.

In order to fit Extra Tree models, we use Scikit-learn package:

```
from sklearn.ensemble import ExtraTreesClassifier
clfExtra = ExtraTreeClassifier(n_estimators = 30, max_depth = 20,
                               min_samples_split = 2, random_state = 0)
```

2.7 Experiment

During our experiment, we tried different models for the predictions, but the highest accurate models we tried are FNN, Random Forest and Extra trees. Also, we found that in order to produce a low MAPE, the number of features left for training after PCA reduction varies from each dataset. We also found that there is a relationship between the number of trees used in the random forest classifier algorithm and the number of components will be used after the PCA. With a certain number of trees, the number of features after PCA reduction has a certain range that will result a relative low MAPE. At the same time, with larger number of trees, this range will go up. However there is one thing to be sure that is the number of components after the PCA reduction should not be too high, it tends to fit poorly as the effect of most features in the 7200 features are negligible.

As the random forest will generate trees using random selected subsets of training samples, the MAPE is not very stable every time we run the algorithm, and the range to choose from is too large, we are not able to determine what will be the best combination of the number of trees, and the number of features to train. We will choose the 'best' combination by running them through a reasonable range, and each will be repeated for 29 trials.

```
pca = PCA(n_components=53)
pca.fit(X_train)
X_train_pca = pca.transform(X_train)
X_train_pca = X_train_pca.tolist()
for i in range(1,30):
    clf1.fit(X_train_pca, Y_train)
    X_test_pca = pca.transform(X_test)
    y_val_pred = clf1.predict(X_test_pca)
    y_out += Range_train[y_val_pred.astype(np.int32)-1]
    print(i)

y_out /=29
```

3. Results

3.1 MAPE of FNN

Frequency	No. of Hidden neurons	No. of PCA	Data1	Data3	Data4	Data5
350Hz	1024	20	90.46	36.65	None	None
950Hz	1024	20	83.31	36.99	None	None
300-950Hz Δf=10	1024	20	5.09	2.84	1.50	10.74
300-950Hz Δf=10	1024	10	5.12	3.67	1.84	5.45
300-950Hz Δf=10	1024	100	12.28	4.13	1.24	6.67
300-950Hz Δf=10	64	20	28.31	5.05	1.39	2.18

300-950Hz Δf=10	128	20	21.45	29.95	1.45	27.63
300-950Hz Δf=10	256	20	6.12	22.12	1.27	16.44
300-950Hz Δf=10	512	20	9.93	2.15	1.42	20.82
300-950Hz Δf=10	2048	20	5.35	2.48	1.38	2.06

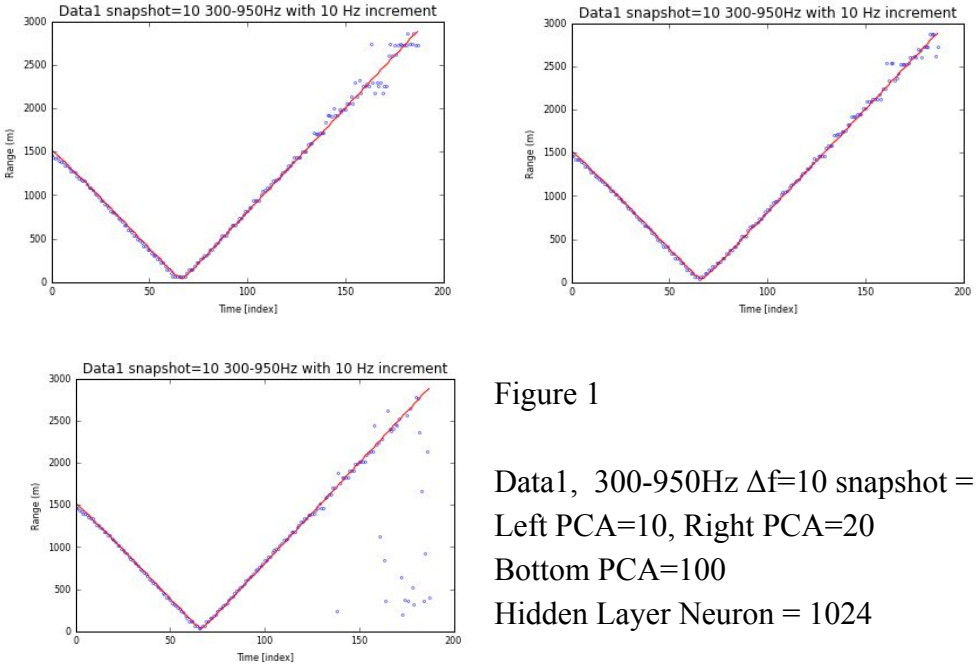


Figure 1

Data1, 300-950Hz Δf=10 snapshot = 10
Left PCA=10, Right PCA=20
Bottom PCA=100
Hidden Layer Neuron = 1024

For dataset 1 with hidden layer neurons equal to 1024, when PCA=20, it gives the best MAPE. Principal component analysis significantly reduce the dimension of the original data but still keep the feature with largest variance. The original dataset with 300-950Hz and 10 increments has over 10000 features. It is much larger than the number of training data. So Preprocessing the data with PCA can help FNN to capture the essence of the training data. When PCA is too low it will make data lose original information. Retaining too many features will disturb the ability of FNN to learn the pattern of data. In this particular case, when PCA is 20 it gives better result especially when the source are far from the sensors. (See figure 1)

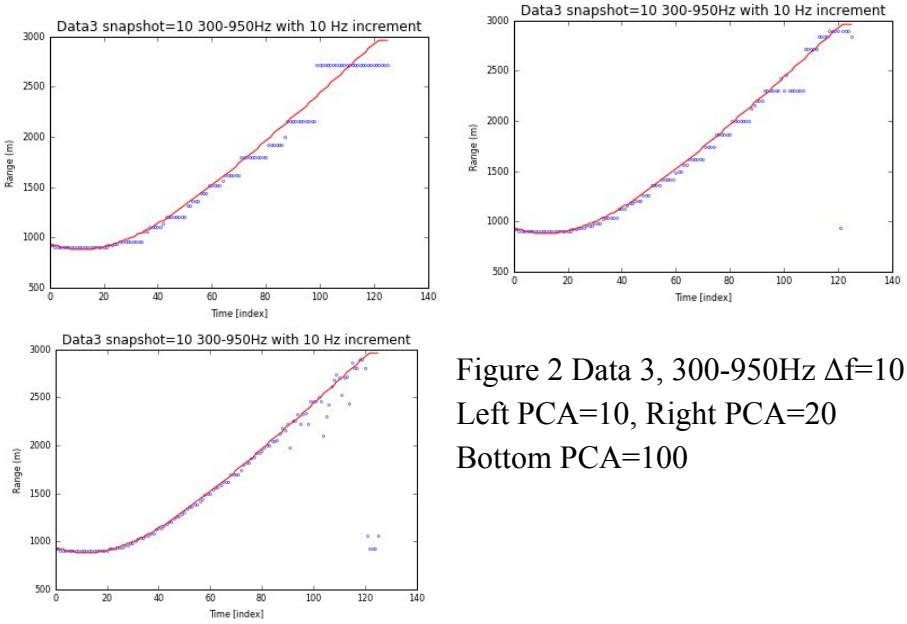


Figure 2 Data 3, 300-950Hz Δf=10 snapshot = 10
Left PCA=10, Right PCA=20
Bottom PCA=100

PCA=20 may not be the best for each dataset, it still gives fairly good result. We also notice that MAPE is not always a good tool for measuring the FNN to predict the location. From figure 2, even PCA = 100 gives higher MAPE compared to PCA = 20, it has a better visual result as the predicted location are distributed along the ground truth curve while PCA =20 gives a prediction looks like stairs. The outliers of PCA=100 affect the accuracy, but prediction the continuous movement can neglect the outliers since the object cannot jump in a sudden.

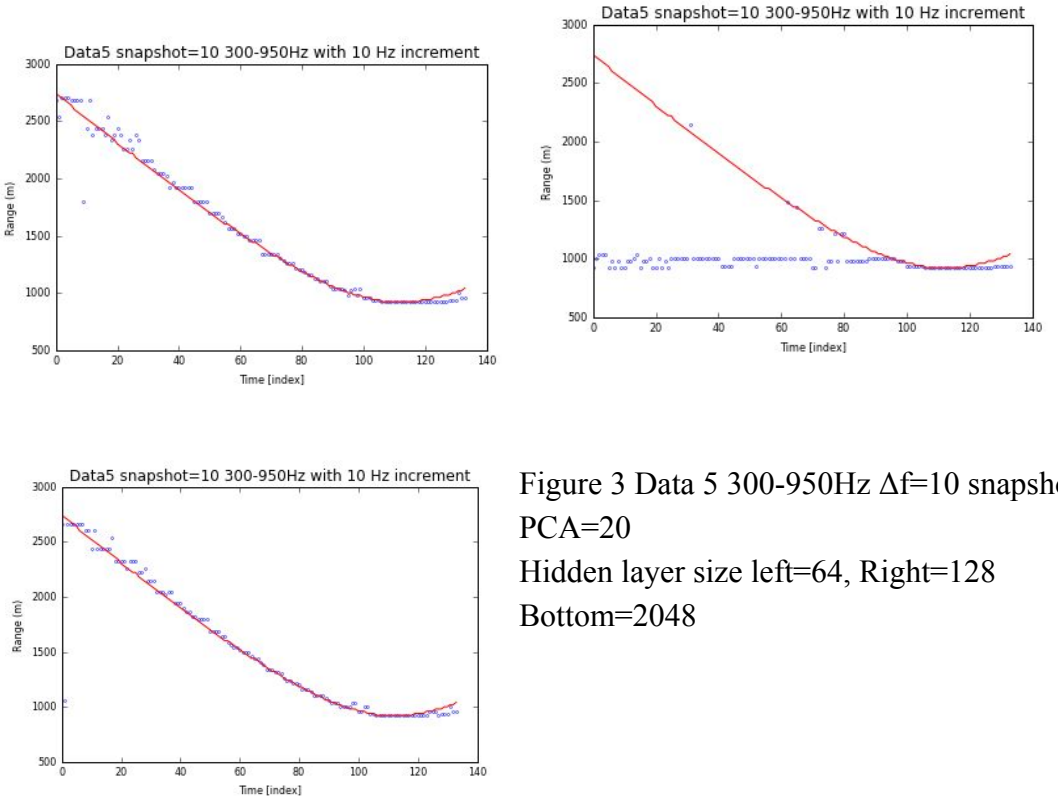


Figure 3 Data 5 300-950Hz Δf=10 snapshot = 10
PCA=20
Hidden layer size left=64, Right=128
Bottom=2048

The number of hidden layer neurons may affect the result significantly. From the figure above, for datasets 5, when the size is 64, it gives 2.18 MAPE. But when increasing the size to 128 the FNN will fail completely. Only when the size reaches 1024 will it give acceptable result. For dataset 3, size 64 will lead to a complete fail. However, datasets 4 are immune to the change of size of the hidden layer neurons.We found that 1024 neurons is generally a good size for datasets. Depending on different datasets the optimal choice of hidden layer size is different.

3.2 MAPE of RF and ET

We compared the predictions before and after preprocessing the data using PCA dimension reductions

Comparison of algorithm with and without PCA

Random Forest:

dataset 01:

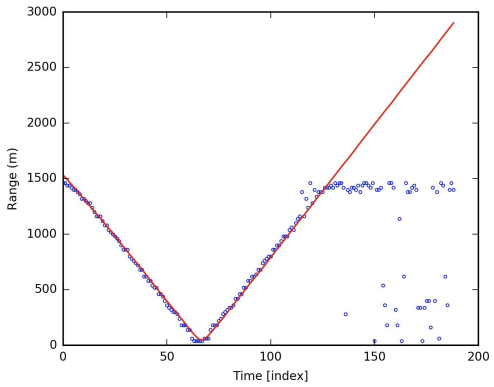


Figure 1 w/o PCA

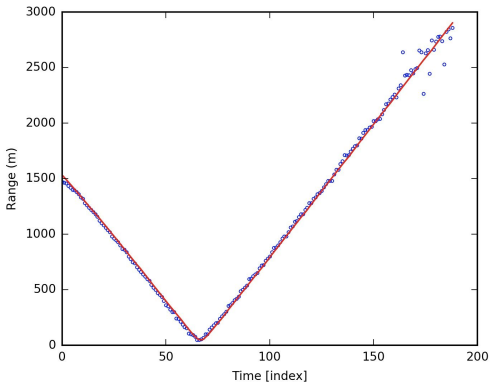


Figure 2 w/ PCA=53

Figure 1 is the prediction using the random Forest after Processing PCA dimension reduction. The model made a good prediction for around first 130 data. However, the differences between the real data and predictions are large after the first 130 data. So the MAPE for Figure 1 is only 183.86%. After the PCA reduction, the differences between the real data and the predictions becomes much smaller. The MAPE for figure 2 is 3.65%.

Extra Tree:

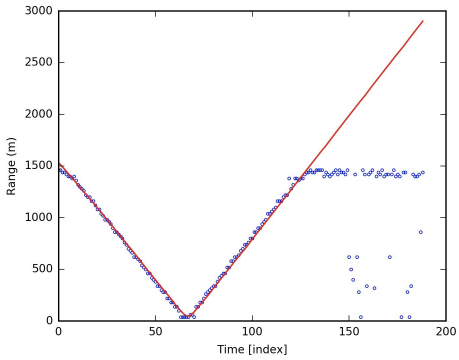


Figure3 w/o PCA

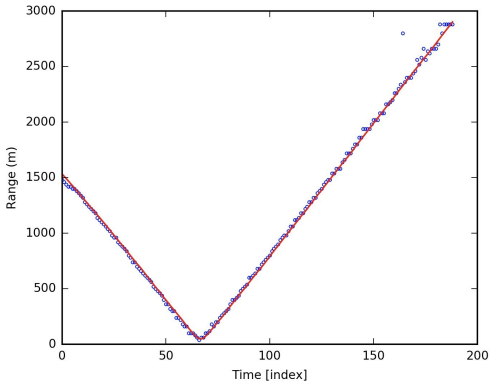


Figure 4 w/PCA=53

Figure 3 is the predictions using Extra trees after processing PCA dimension reduction. As we could see that the ability of the model for predictions becomes much weaker after first 130 data, similar as the random forest algorithm. The MAPE for this model is 142.2%. However, after the PCA reduction, the prediction describes the real data with minor mistakes. The MAPE for this model is 3.44%

Comparison of Random Forest and Extra Tree

Since Random Forest and Extra Tree are really similar prediction models, we compared the result predicted by the two models.

dataset 03:

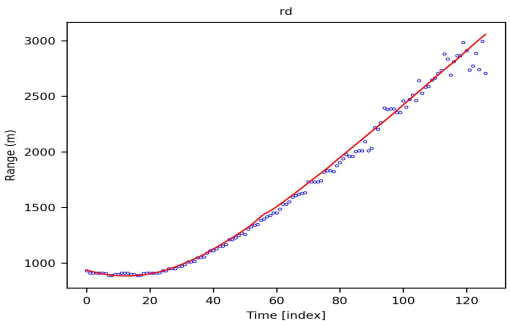


Figure 5(Random Forest) w/ PCA=20

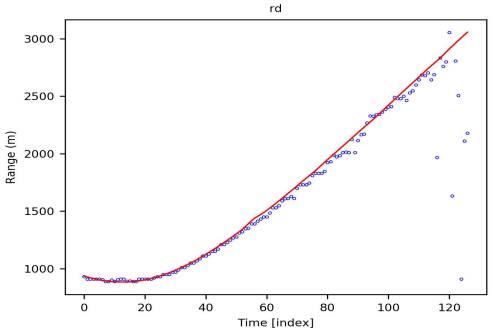
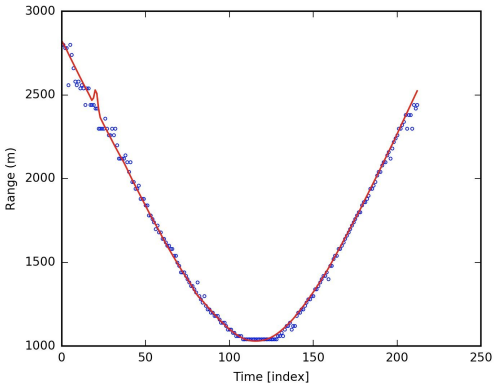


Figure 6(Extra Tree) w/ PCA=20

After we pre processed the data, the MAPE in random forest model is 1.9%. The MAPE in Extra Tree is 4.6%. It is easily to see from the figure 6 that there are some really bad predictions after the first 120 points.

dataset 04:



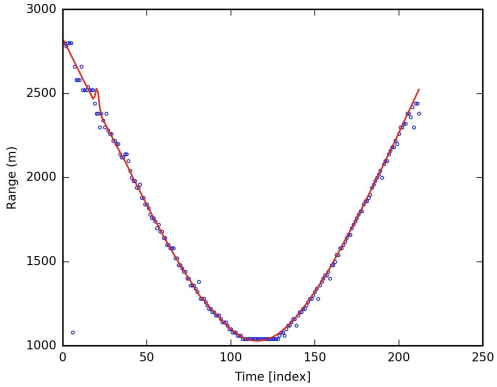


Figure7(Random Forest) w/ PCA=34 Figure8(Extra Tree) w/ PCA=34

After pre-processed the data, the MAPE in random forest is 1.12% but the MAPE in Extra Tree is 1.04%. Though both models performed the data well, Extra Tree has lower MAPE in this dataset.

Based on 5 datasets using both Random Forest and Extra Tree classifiers, they both have better performance on some datasets.

4. Conclusion

Without using Principle Component Analysis to form reduced number of features for training, FNN overperformed Random Forest and Extra Tree in terms of MAPE.

By using PCA dimension reduction, all three methods were able to achieve high accuracy by using a small number of features instead of the original 7200 features. The accuracy of the results could be highly influenced by the number of dimensions(features) in train data. There does not exist a large difference between the performance of FNN, RF and ET over the given 5 datasets with proper tuning parameters.

5. Reference

[1] Haiqiang Niu, Emma Reeves,and Peter Gerstoft. *Source localization in an ocean waveguide using supervised machine learning. (2017)*. Scripps Institution of Oceanography, University of California San Diego, La Jolla, California 92093-0238, USA

[2] C. M. Bishop(2006), Pattern Recognition and Machine Learning. Springer, Chaps. 4, 5 and 7.

[3]L. Breiman(2001), “Random Forest”, Mach. Learn. 45, 5-32.

[4]I. Jolliffe (2002). Principal component analysis. Springer. 2nd edition.

[5]C. Ding, X.F. He(2004), Proceedings of the 21 st International Conference on Machine Learning, Banff, Canada

[6]F.F.Li (2008), COS 429 Course material, Princeton University, retrieved from: http://www.cs.princeton.edu/courses/archive/fall08/cos429/CourseMaterials/lecture2/PCA_handout.pdf

[7]Geurts, Pierre, Damien Ernst, and Louis Wehenkel. "Extremely randomized trees." Machine Learning 63.1 (2006): 3-42. Web.