A Comparative Analysis of Machine Learning Techniques

P. A. Ozoh[1], A. A. Adigun[2], L. O. Omotosho[3]
[1,2,3]*Department of ICT, Osun State University, Osogbo, Nigeria*

***Abstract*- Machine learning is a branch of artificial intelligence that is used to analyze large set of data. Machine learning approach is a statistical approach on learning more about a raw data set. When considering the existing systems in the world, there is a huge output of data which are not well analyzed. The use of machine learning techniques provide a way of analyzing a huge data set in order to find patterns and relationships among different entities which cannot be observed without advanced analyzing techniques. In this paper, the machine learning techniques that will be considered include; Box-Jenkins method, artificial neural network (ANN) technique, and Kalman technique. Each technique will be implemented using python, and the results obtained using the mentioned methods will be compared. This paper explores the application of effective machine learning to overcome challenges associated with data analysis and demonstrates how machine learning techniques have contributed and are contributing to research in machine learning.**

***Keywords:* machine learning, big data, Box-Jenkins, artificial neural network, kalman technique**

I. INTRODUCTION

In recent years, the amount of information that can be extracted from data has rapidly increased. Machine learning is not just about storing large amounts of data, but it is a part of artificial intelligence (AI). Artificial Intelligence is the improvement of the computer programs to perform tasks that usually require the human intervention, such as decision making (Mohamed, 2017). The field of machine learning is concerned with how to construct computer programs that automatically improves with experience. Each machine learning method has its strengths and limitations, and as real world problems do not always satisfy the assumptions of a particular method, one approach is to apply an appropriate machine learning technique and select the one that provides the best solution (Alesheykh, 2016). This research explores the application of effective machine learning to overcome challenges associated with data analysis and demonstrates how machine learning techniques have contributed to research.

As an intelligent system, machine learning techniques can be used to understand the meaning of a data set in a logical way and provide useful outputs from raw data for different purposes. Machine learning approach is a statistical approach on learning more about a raw data set. When considering the existing systems in the world, there is a huge output of data which are not well analyzed. The machine learning techniques provide a way of analyzing a huge data set in order to find patterns and relationships among different entities which cannot be observed without advanced analyzing techniques (Vibatha et al., 2016).In this research, some machine learning techniques such as Box-Jenkins method, artificial neural network, and Kalman technique are implemented using python and the results obtained from the mentioned methods are compared.

The contributions of this research are as follows:

1. This research will be able to identify a reliable machine learning technique
2. This research will be applicable to big data
3. The application of the machine learning techniques will help to push the accuracy of models beyond those achieved by conventional techniques

II. LITERATURE REVIEW

Machine learning is one of the important lanes of artificial intelligence which is a very important subject in the research or industry. Machine learning, like artificial intelligence covers a broad range of processes that it is difficult to define precisely. Machine learns whenever it changes its structure, program, or data (based on its inputs or in response to external information) in such a manner that its expected future performance improves. Some of these changes, such as the addition of a record to a data base, falls comfortably within the province of other disciplines and are not necessarily better understood for being called learning. Machine learning usually refers to the changes in systems that perform tasks associated with artificial intelligence (AI). Such tasks involve recognition, diagnosis, planning, robot control, prediction, etc (Nils, 2005). The essence of machine learning is to compile data when the program learns to generate the information that be used. Machine learning is a field of computer science that uses statistical techniques to give computer systems the ability to learn (e.g., progressively improve performance on a specific task) with data, without being explicitly programmed (Samuel, 1959). It refers to the automated detection of meaningful patterns in data.

The perceptron was introduced (Rosenblatt, 1958). The perceptron is designed to illustrate some of the fundamental properties of intelligent systems in general, without becoming too deeply enmeshed in the special and frequently unknown conditions which hold for particular biological organisms. (Minsky & Papert, 1969) proposed the famous XOR problem. Thereafter, work on neural network researches were dormant up until 1980s. Neural network researchers successively presented the idea of multi-layer perceptron with practical back-propagation training (Rumelhart, 1985). Back-propagation is the key to neural network architectures. One of the most important machine learning breakthroughs was the support vector machines (SVM), proposed by Cortes&Vladimir (1995), with very strong theoretical standing and empirical results. Support vector machines are able to exploit all the knowledge of convex optimization, generalization margin theory and kernels.

A machine learning model was proposed by Freund et al. (1999). It proposeda set of classifiers called Adaboost. Adaboost set of classifiers are easy to train, and this model is the basis of many different tasks like face recognition and detection. The integrated grey model with multiple regression model (IGMMRM) was applied to modeling of data, in comparison with Grey model (GM) and multiple regression method (Wang & Xia. 2009). The modeling techniques were assessed using *relative error* (RE), *mean absolute error* (*MAE*), *root mean square error* (*RMSE*), and *mean absolute percentage error* (*MAPE*) to evaluate the accuracy of the models. The study suggests that the performance of IGMMRM was higher than the other two models based on historical data.

Artificial neural network (ANN) was described in Damak (2011) as a hidden-layer feed-forward network technique and it's a widely used technique for time-series modelling and forecasting. The paper described that the technique is based on pattern recognition, and able to forecast for non-linear models. Neural networks are similar to the least square estimation technique and can be viewed as an alternative statistical approach to solving least squares problems (Chen et al. 2001). The paper presented an artificial neural network-based short-term load forecasting technique for estimating data. The ANN technique utilized a combination of the three layer feed-forward neural network and a back-propagation training technique.

Computational analysis is a collection of procedures that is used to process large amounts of data with a view of obtaining results based on processed data and as a result, getting their behavioral pattern. A review of machine learning techniques is undertaken, with the aim of identifying and selecting an accurate and reliable technique for modelling data. The techniques are discussed in the following sections.

A. Box-Jenkins Technique

The general form of the model that is used to describe Box-Jenkins technique is

$$\varphi(B)\mathrm{z_t} = \phi(B)\nabla^{\mathrm{d}}\mathrm{z_t} = \theta_0 + \theta(B)\mathrm{a_t}$$

where

$$\phi(B) = 1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p$$
$$\theta(B) = 1 - \theta_1 B - \theta_2 B^2 - \cdots - \theta_q B^q$$

$\phi(B)$ and $\theta(B)$ are polynomial operators in B of degrees p and q. This process is referred to as an ARMA (p, q) process. The ARIMA model can be expressed explicitly in terms of current and previous shocks (Box et al., 2008). A linear model can be written as the output z_t from the linear filter

$$\begin{aligned} z_t &= a_t + \psi_1 a_{t-1} + \psi_2 a_{t-2} + \cdots \\ &= a_t + \sum_{j=1}^{\infty} \psi_j a_{t-j} \\ &= \psi(B) a_t \end{aligned}$$

whose input is a white noise, or a sequence of uncorrelated shocks a_t with mean 0 and common variance σ_a^2. Then

$$\varphi(B) z_t = \varphi(B)\psi(B) a_t$$

B. Artificial Neural Network

Artificial neural network (ANN) is based on the recognition that the human brain computes in an entirely different way from the conventional digital computer. It has the capability to organize its structural constituents, known as neurons, so as to perform certain computations (e.g., pattern recognition, perception, and motor control) many times faster than the fastest digital computer in existence today(Simon, 2009).

The neural network has at least two physical components, namely, the processing elements and the connections between them. The processing elements are called neurons, and the connections between the neurons are known as links. Each neuron receives stimulus from the neighboring neurons connected to it, process the information, and produces an output. Neurons that receive stimuli from outside the network are called input neurons. Neurons whose outputs are used externally are called output neurons. Neurons that receive stimuli from other neurons and whose output is a stimulus for other neurons in the neural network are known as hidden neurons (David, 2005).The block diagram of Fig. 1 shows the model of a neuron, which forms the basis for designing a large family of neural networks.

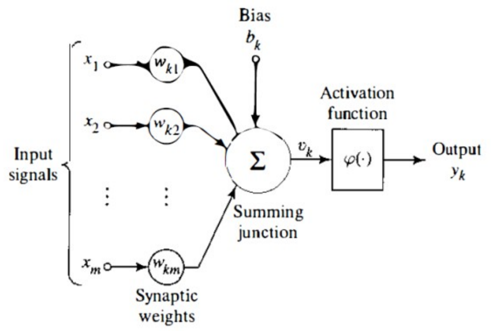


Fig.1 Artificial neuron model (Simon, 2009)

The three basic elements of the neural model are:

1) A set of synapses, or connecting links, each of which is characterized by a weight or strength of its own. Specifically, a signal x_j at the input of synapse j connected to neuron k is multiplied by the synaptic weight

ω_{kj}. It is important to make a note of the manner in which the subscripts of the synaptic weight ω_{kj} are written. The first subscript in ω_{kj} refers to the neuron in question, and the second subscript refers to the input end of the synapse to which the weight refers. Unlike the weight of a synapse in the brain, the synaptic weight of an artificial neuron may lie in a range that includes negative as well as positive values.

2) An adder for summing the input signals, weighted by the respective synaptic strengths of the neuron; the operations described here constitutes a linear combiner.
3) An activation function for limiting the amplitude of the output of a neuron. The activation function is also referred to as a squashing function, in that it squashes (limits) the permissible amplitude range of the output signal to some finite value (Simon, 2009).

Artificial neural network with feed-forward topology is called feed-forward artificial neural network and as such has only one condition: information must flow from input to output in only one direction with no back-loops. There are no limitations on number of layers, type of transfer function used in individual artificial neuron or number of connections between individual artificial neurons (Andrej, 2011). This is presented in Fig. 2.

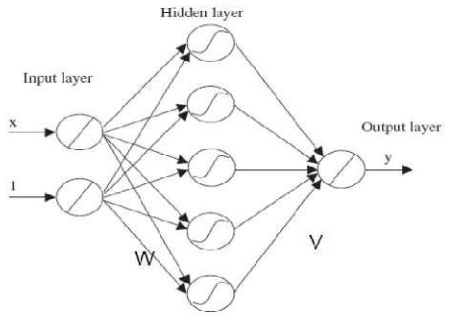


Fig.2Feed-forward network

C. Kalman Filter Adaptation Algorithm

The Kalman filter addresses the general problem of trying to estimate the state $x \in \mathfrak{R}^n$ of a discrete-time controlled process that is governed by the linear stochastic difference equation

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1}$$

with a measurement $z \in \mathfrak{R}^n$ that is

$$z_k = Hx_k + v_k$$

The random variables w_k and v_k represent the process and measurement noise, respectively (Welch & Bishop, 2006). They are assumed to be independent of each other and with normal probability distributions

$$p(w) \sim N(0, Q)$$

$$p(v) \sim N(0, R)$$

The a priori and *a* posterior estimates are defined as:

$$e_k^- \equiv x_k - \hat{x}_k^- \quad \text{, and}$$

$$e_k \equiv x_k - \hat{x}_k$$

The a priori estimate error covariance is then

$$P_k^- = E[e_k^- e_k^{-T}]$$

and the a posteriori estimate error covariance is

$$P_k = E[e_k e_k^T]$$

In deriving the equations for the Kalman filter, an equation is derived that computes an a posteriori estimate $\hat{x}_k$ as a linear combination of an a priori estimate $\hat{x}_k^-$ and a weighted difference between an actual measurement z_k and a measurement prediction Hx_k^-.

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$$

The resulting K is given byMaybeck (1979); Brown and Hwang (1992); Jacobs (1993)

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$$

$$= \frac{P_k^- H^T}{HP_k^- H^T + R}$$

The equations for the Kalman filter falls into two groups: time update equations and measurement update equations. The time update equations are responsible for projecting forward in time the current state and error covariance estimates to obtain the a priori estimates for the next time step. The measurement update equations are responsible for the feedback for incorporating a new measurement into the a priori estimate to obtain an improved *a* posteriori estimate. The time update equations can be defined as predictor equations, while the measurement update equations are defined as corrector equations.

The time update projects the current state estimate ahead in time. The measurement update adjusts the projected estimate by an actual measurement at that time. The discrete Kalman filter time update equations are (Welch & Bishop, 2008):

$$\hat{x}_k^- = A\hat{x}_{k-1}^- + Bu_{k-1}$$

$$P_k^- = AP_{k-1}A^T + Q$$

The discrete Kalman filter measurements update equations are (Welch & Bishop, 2008):

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$$

$$P_k = (I - K_k H)P_k^-$$

The first task during the measurement update is to compute Kalman gain, K_k. The next step is to actually measure the process to obtain z_k, and then to generate an a posteriori state estimate. The final step is to obtain an a posteriori error covariance estimate. After each time and measurement update pair, the process is repeated with the previous a posterior estimates used to predict the new a priori estimates. This recursive nature is one of the very appealing features of the Kalman filter.

III. METHODOLOGY

This section gives an outline of research methods that are used in this study. The overall objective of this research is to perform a comparative analysis of machine learning techniques. The system development and design of this model for comparative analysis of machine learning techniques was undertaken by implementing the various machine learning algorithms using Python programming language.The comparative analysis model input data set into the Box-Jenkins, ANN, and Kalman technique, then a testing dataset is also passed into this machine learning technique to test the ability of these algorithms and then the output is used as an evaluative mechanism for performance comparison. This is shown in Fig. 3.

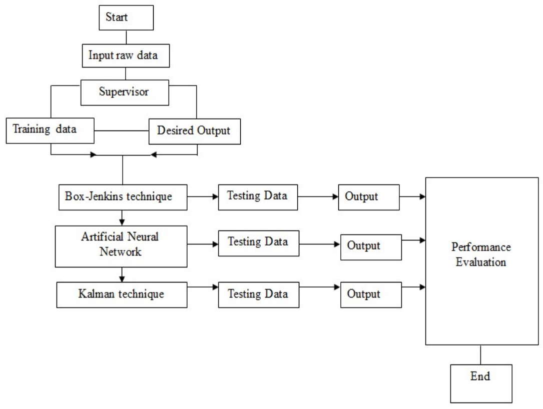


Fig.3Comparative analysis model

A. Data Set

The research utilizes an energy distribution and usage dataset collected online. It has the following fields: The dataset has the following fields: Date, Time, Global_active_power,Global_reactive_power,Voltage,Global_intensity, Sub_metering_1,Sub_metering_2,Sub_metering_3.

The dataset has 260,641 data value, where 154,261 were used for training the machine learning techniques, while 56,924 was used as the testing dataset leaving 49,456.

The algorithms for the Box-Jenkins, artificial neural network and Kalman techniques are tested on an energy distribution and usage dataset.

B. Implementation of Machine Learning Techniques

The implementation of the study requires the analysis of the machine learning algorithms for the comparative analysis of machine learning techniques. The graphical analytics performance of the techniques are discussed in the following sections.

Box-Jenkins Technique: The system design of the Box-Jenkins technique is given on Fig. 4.

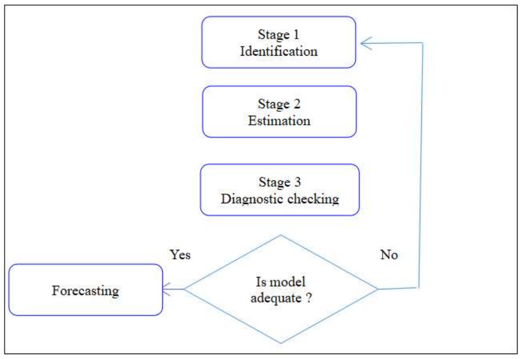


Figure 4 Data flow diagram for Box-Jenkins

The algorithm for Box- Jenkins algorithm is given as follows:

Step 1

Define significance level α, and estimate σ using confidence interval. Set l = 0.

Step 2

If l is outside the confidence interval (,), set l = l+1, and repeat Step 2

Else, go to Step 3

Step 3

Calculate the percentage φ of auto correlation coefficients from lag l to the maximum lag.

If φ is less than or equal to 1-α,

then let q=l and stop.

Else, q is undetermined and stops.

Comments are in order on this algorithm.

If q= 0 from the above algorithm,

then the residual can be said to be a white noise.

Box-Jenkins technique was implemented using python programming language (Appendix A). The output is displayed in Fig. 5 and Fig. 6.

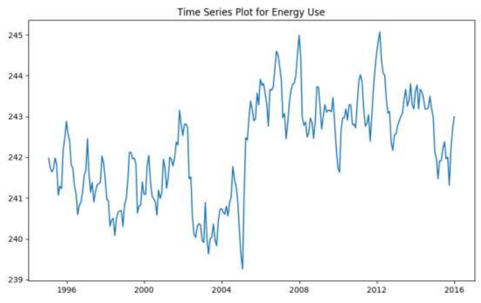


Fig.5 Implementation diagram for Box-Jenkins

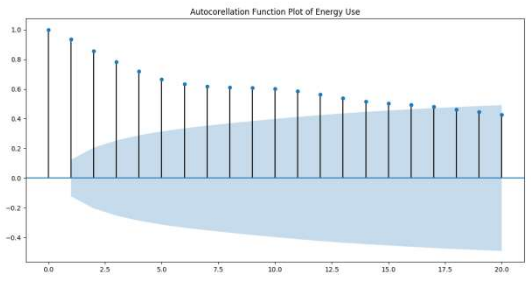
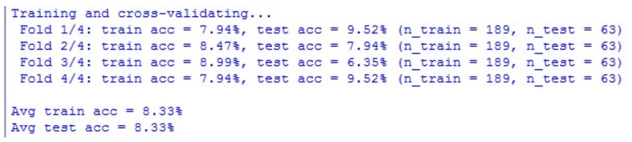


Fig.6Auto correlation plot of energy

Artificial Neural Network Technique: This study uses the artificial neural network back propagation method, and implemented using python. The programme is presented in Appendix B. The output is displayed in Figure 7.



```
Training and cross-validating...
 Fold 1/4: train acc = 7.94%, test acc = 9.52% (n_train = 189, n_test = 63)
 Fold 2/4: train acc = 8.47%, test acc = 7.94% (n_train = 189, n_test = 63)
 Fold 3/4: train acc = 8.99%, test acc = 6.35% (n_train = 189, n_test = 63)
 Fold 4/4: train acc = 7.94%, test acc = 9.52% (n_train = 189, n_test = 63)

Avg train acc = 8.33%
Avg test acc = 8.33%
```

Fig.7 Implementation for artificial neural networks

Kalman Filter Adaptation Technique: The output using Kalman filter algorithm and implemented by python programming language is given in Figure 8. The Program is displayed in Appendix C.

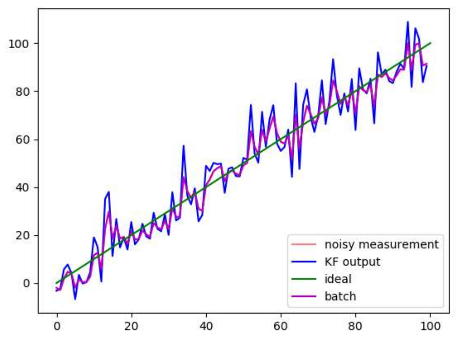


Fig.8Implementation for Kalman filter

IV. SUMMARY AND CONCLUSION

This research aims to do a comparative analysis of machine learning techniques. The study was tested and implemented using python programming language on Box-jenkins, ANN, amd Kalman filter techniques, and the results were presented in a graphical form. Performance evaluation were done to compute graphically the overall result of the proposed model on machine learning techniques. The three machine learning techniques were described. Also presented were their objected oriented based models. The modelswere used to evaluate the performance of these three machine learning technique. Based on this result, it clearly shows the complexity of the algorithms and how well they work on data making ANN as the highest rated.

REFERENCES

[1] Alesheykh, R., (2016). Comparative Analysis of Machine Learning Algorithms with Optimization Purposes. Control and Optimization in Applied Mathematics (COAM) Vol. 1, No. 2, pp. 63-75.

[2] Mohamed, A., (2017). Comparative Study of Four Supervised Machine Learning Techniques for Classification. International Journal of Applied Science and Technology.

[3] Chen, X., Xu, L., Yao, Q.,(2014). Study of a Distribution Line Overload Control Strategy Considering the Demand Response. Electric Power Components and Systems, Vol. 42, No. 9, pp. 970–983.

[4] Cortes, C., Vladimir, V., (1995). Support-vector networks. Machine learning Vol. 20, No. 3, pp. 273-297.

[5] Damak, S., (2011). Applications of Two Identification Methods For an Electric Distribution System. Journal of Automation & Systems Engineering, Vol. 4, pp. 176–184.

[6] Freund, S., Robert, N., (1999). A Short Introduction to Boosting. Journal-Japanese Society for Artificial Intelligence. pp. 771-780.

[7] Minsky, M., Papert, S., (1969). Perceptrons, M.I.T. Press.

[8] Nils. L., (2005)., Introduction to Machine Learning.Department of Computer Science Stanford University. Stanford, CA 94305.

[9] Ozoh, P., Olayiwola, M.,Ogundoyin, I., (2020). Analysis of Computational Techniques to Analyze Big Data, Communications in Applied Sciences, Vol. 8, No. 1, pp. 1-18.

[10] Rosenblatt, A., (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, Psychological Review, Vol.65, No. 6,pp. 386-408.

[11] Rumelhart, E.,Geoffrey, J., Ronald W., (1985). Learning internal representations by error propagation. No. ICS-8506. California Univ San Diego La Jolla Institute for Cognitive Science.

[12] Samuel, A., (1959). Some Studies in Machine Learning Using the Game of Checkers. IBM Journal of Research and Development.

[13] Vibhatha, K., Nishadi, S., Anuruddha, R., Pasika, U., (2016). Electrical Devices Identification through Power Consumption using Machine Learning Techniques. Researchgate. doi:10.5013/IJSSST.a.17.32.13.1473-8031.

[14] Wang, F., Xia. X., (2009). Integration of Grey Model and Multiple Regression Model toPredict Energy Consumption,IEEE Proceedings, pp. 194–197.

APPENDIX A

Python Code for Box-Jenkins Technique

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.tsa.arima_model import ARIMA, ARIMAResults
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import matplotlib.pylab as plt
import matplotlib.dates as dates
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 15, 6
######################################################
        LOAD DATA
######################################################
loading data
dataMaster = pd.read_csv('dataset.csv')
sp_500 = dataMaster['Voltage']
print(sp_500.head(12))
```

```
ran = pd.date_range('1995-01', '2016-1', freq = 'M')
ts = pd.Series(dataMaster['Voltage'].values, index = ran)
print(ts.head(12))
print(ts.dtypes)
#####################################################
##
            DO EXPLORATORY ANALYSIS
#####################################################
##
plt.plot(ts)
plt.title('Time Series Plot for Energy Use')
 plt.xlim([0, 255])
plt.show()

sp500_TR = ts['1995':'2014']
print(sp500_TR)
#####################################################
##
            MODEL ESTIMATION
#####################################################
##
DIAGNOSING ACF
acf = plot_acf(ts, lags = 20)
plt.title("Autocorellation Function Plot of Energy Use")
acf.show()
# DIAGNOSING PACF
pacf = plot_pacf(ts, lags = 20)
plt.title("Partial Autocorellation Function Plot of Energy
Use")
pacf.show()
 TRANSFORMING OUR DATA TO ADJUST FOR
NON-STATIONARITY
sp500_diff = ts - ts.shift()
diff = sp500_diff.dropna()
print(diff.head(12))
print(diff.dtypes)
plt.figure()
plt.plot(diff)
plt.title('First Difference Time Series Plot')
plt.show()
acfDiff = plot_acf(diff, lags = 20)
plt.title("ACF Plot of Energy Use(Difference)")
acfDiff.savefig("images/timeSeriesACFDiff.png", format =
'png')
acfDiff.show()
edit this shit on the actual project !
pacfDiff = plot_pacf(diff, lags = 20)
plt.title("PACF Plot of Energy Use(Difference)")
pacfDiff.savefig("images/pacfDiff.png", format = 'png')
pacfDiff.show()
#####################################################
##
            BUILD MODEL
#####################################################
##
mod = ARIMA(sp500_TR, order = (0, 1, 1), freq = 'M')
results = mod.fit()
print(results.summary())
#####################################################
##
            FORECAST
#####################################################
##
predVals = results.predict(239, 251, typ='levels')
print(predVals)
predVals = predVals.drop(predVals.index[0])
print(predVals)
sp500_for = pd.concat([ts, predVals], axis = 1,
keys=['original', 'predicted'])
print(sp500_for['2014':'2015'])
plt.figure()
plt.plot(sp500_for)
plt.title("Actual Vs. Forecasted Values")
plt.savefig("images/sp500_for.png", format = 'png')
plt.show()
plt.figure()
plt.plot(sp500_for)
plt.title('Real Vs. Predicted Values for 2015')
plt.savefig("images/sp500_for2.png", format = 'png')
plt.show()
```

APPENDIX B

Python Code for Artificial Neural Network

```
import numpy as np
from src.NeuralNetworkClass import NeuralNetwork
import src.utils as utils
def main():
    # ===================================
    # Settings
    # ===================================
    filename = "data/dataset.csv"
    n_hidden_nodes = [5]  # nodes in hidden layers i.e.
[n_nodes_1, n_nodes_2, ...]
    l_rate = 0.6   # learning rate
    n_epochs = 800   # number of training epochs
    n_folds = 4   # number of folds for cross-validation
    print("Neural network model:\n n_hidden_nodes =
{}".format(n_hidden_nodes))
    print(" l_rate = {}".format(l_rate))
    print(" n_epochs = {}".format(n_epochs))
    print(" n_folds = {}".format(n_folds))
    # ===================================
    # Read data (X,y) and normalize X
    # ===================================
    print("\nReading '{}'...".format(filename))
    X, y = utils.read_csv(filename)   # read as matrix of
floats and int
    utils.normalize(X)   # normalize
    N, d = X.shape   # extract shape of X
    n_classes = len(np.unique(y))
    print(" X.shape = {}".format(X.shape))
    print(" y.shape = {}".format(y.shape))
    print(" n_classes = {}".format(n_classes))
    # ===================================
    # Create cross-validation folds
    # These are a list of a list of indices for each fold
    # ===================================
    idx_all = np.arange(0, N)
    idx_folds = utils.crossval_folds(N, n_folds, seed=1)
    # ===================================
    # Train and evaluate the model on each fold
    # ===================================
    acc_train, acc_test = list(), list()    # training/test accuracy
score
    print("\nTraining and cross-validating...")
    for i, idx_test in enumerate(idx_folds):
        # Collect training and test data from folds
        idx_train = np.delete(idx_all, idx_test)
        X_train, y_train = X[idx_train], y[idx_train]
        X_test, y_test = X[idx_test], y[idx_test]
        # Build neural network classifier model and train
        model = NeuralNetwork(n_input=d,
n_output=n_classes, n_hidden_nodes=n_hidden_nodes)
        model.train(X_train, y_train, l_rate=l_rate,
n_epochs=n_epochs)
        # Make predictions for training and test data
        y_train_predict = model.predict(X_train)
        y_test_predict = model.predict(X_test)
        # Compute training/test accuracy score from
predicted values

acc_train.append(100*np.sum(y_train==y_train_predict)/len(
y_train))
acc_test.append(100*np.sum(y_test==y_test_predict)/len(y_te
st))

        # Print cross-validation result
        print(" Fold {}/{}: train acc = {:.2f}%, test acc =
{:.2f}% (n_train = {}, n_test = {})".format(i+1, n_folds,
acc_train[-1], acc_test[-1], len(X_train), len(X_test)))

    # ===================================
    # Print results
    # ===================================
    print("\nAvg train acc =
{:.2f}%".format(sum(acc_train)/float(len(acc_train))))
    print("Avg test acc =
{:.2f}%".format(sum(acc_test)/float(len(acc_test))))
# Driver
if __name__ == "__main__":
    main()
```

APPENDIX C

Python Code for Kalman Filter Technique

```
from __future__ import (absolute_import, division,
print_function,
                        unicode_literals)
import numpy.random as random
import numpy as np
import matplotlib.pyplot as plt
from filterpy.kalman import FadingKalmanFilter
from pytest import approx
from scipy.spatial.distance import mahalanobis as
scipy_mahalanobis
DO_PLOT = False
def test_noisy_1d():
    f = FadingKalmanFilter(3., dim_x=2, dim_z=1)
    f.x = np.array([[2.],
                    [0.]])        # initial state (location and
velocity)
    f.F = np.array([[1.,1.],
                    [0.,1.]])    # state transition matrix
    f.H = np.array([[1.,0.]])    # Measurement function
    f.P *= 1000.                 # covariance matrix
    f.R = 5.**2                  # state uncertainty
    f.Q = np.array([[0, 0],
                    [0, 0.0001]]) # process uncertainty
    measurements = []
    results = []
    zs = []
    for t in range (100):
        # create measurement = t plus white noise
        z = t + random.randn() * np.sqrt(f.R)
        zs.append(z)
        # perform kalman filtering
        f.update(z)
        f.predict()
        # save data
        results.append(f.x[0, 0])
        measurements.append(z)
        # test mahalanobis
        a = np.zeros(f.y.shape)
        maha = scipy_mahalanobis(a, f.y, f.SI)
        assert f.mahalanobis == approx(maha)
        print(z, maha, f.y, f.S)
        assert maha < 4
    # now do a batch run with the stored z values so we can test
that
    # it is working the same as the recursive implementation.
    # give slightly different P so result is slightly different
    f.X = np.array([[2.,0]]).T
    f.P = np.eye(2)*100.
    m, c, _, _ = f.batch_filter(zs,update_first=False)
    # plot data
    if DO_PLOT:
        p1, = plt.plot(measurements,'r', alpha=0.5)
        p2, = plt.plot (results,'b')
        p4, = plt.plot(m[:,0], 'm')
        p3, = plt.plot ([0, 100],[0, 100], 'g') # perfect result
        plt.legend([p1,p2, p3, p4],
                   ["noisy measurement", "KF output",
"ideal", "batch"], loc=4)
        plt.show()
if __name__ == "__main__":
    DO_PLOT = True
```