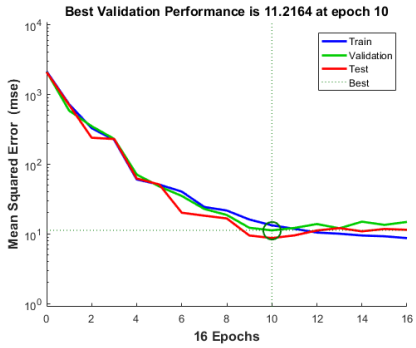
Model Assessment

- Given that most nonlinear models that employ supervised learning have a tendency to over-fit, it is generally best to employ a strategy of partitioning the dataset into training, validation, and testing datasets to ensure good generalization.
- First requirement is to ensure that the dataset offers best possible features to carry out the task at hand. *Feature selection* and *feature extraction* steps might be necessary to improve the performance of the model.
- If dataset size is small, consider employing an *S*-fold (or *K*-fold) cross-validation strategy.
- Ultimate goal is to build a model that best addresses the *bias-variance dilemma* (simpler models have more bias but less variance and vice versa).
- Given a fixed dataset, generally, as one decreases bias, variance increases (and vice versa).
- Building an effective model not only entails selecting the most appropriate modeling approach, but also optimizing the complexity of the model (e.g., in the case of MLPs, the number of hidden layers as well as number of nodes within each layer), parameters associated with the model (e.g., type of transfer function employed by the neurons and their parameters), as well as careful selection of the learning algorithm and its parameters.
- Sensitivity analysis and feature/variable importance ranking might also be critical for some applications.

Function Approximation & Regression

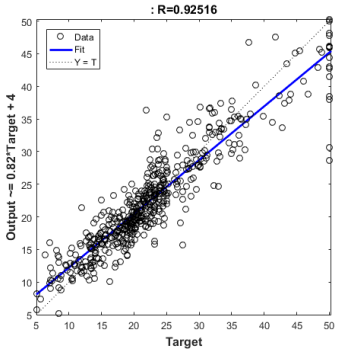
- The performance criterion selected for learning the model should be a complete measure and account for both accuracy and precision (e.g., MSE and MAD are complete measures whereas expected error is incomplete).
- Model errors across training, validation, and testing datasets should be consistent in amplitude.



Sample MSE plot during cross-validation.

```
Matlab:-> [net,tr] = train(net,x,t); plotperform(tr)
```

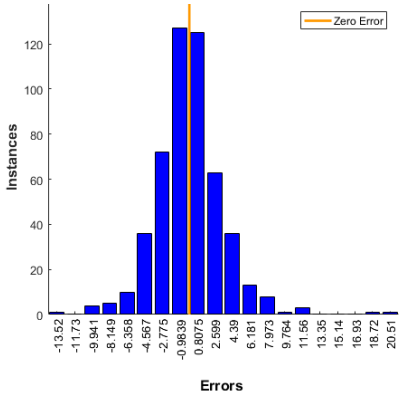
- Another measure of how well the model has fit the data is the regression plot. Do this for training, validation, and testing datasets.



Sample regression plot of actual vs. predicted values.

```
Matlab:-> y = net(x); plotregression(t,y)
```

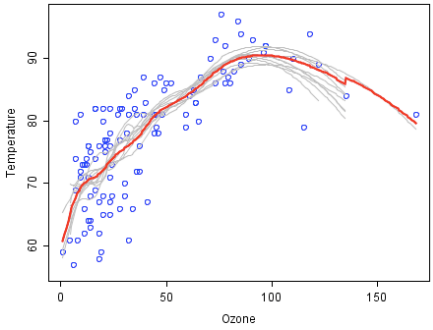
- The regression model should ideally have a high R^2 (a statistical measure of how close the data are to the fitted regression line; is also known as the coefficient of determination; varies between 0-100% and higher values being better) and an intercept of zero and slope of 1.
- Another measure of how well the model has fit data is the error histogram. Typically, most errors should be close to zero, with very few errors far from that. If there are any significant outliers (large errors), these should be investigated further to see if there are data collection errors or if there are missing factors/variables that should be incorporated into the model to reduce the errors.



Sample model residual error histogram.

```
Matlab:-> e = t - y; ploterrhist(e)
```

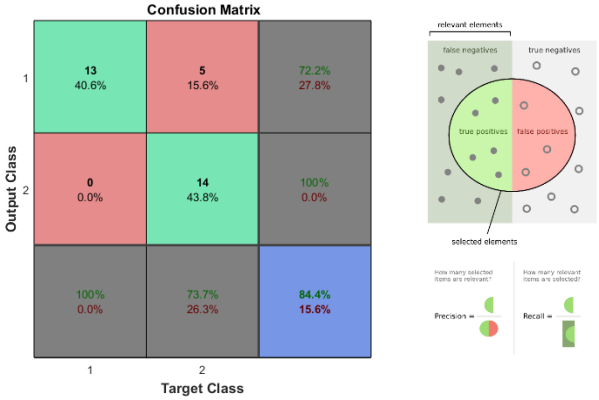
- Given that many models employ learning algorithms that cannot guarantee a global optimal solution during the training cycle, the training process has to be repeated several times to demonstrate that the results are indeed consistent (not exhibiting too much variability).



Sample plot reporting results from different training runs (gray lines).

Pattern Recognition (Classifiers)

- MSE plots are meaningful for classification problems as well.
- A more effective measure is the "confusion matrix". The confusion matrix shows the percentages of correct and incorrect classifications. Correct classifications are the green squares on the matrices diagonal. Incorrect classifications form the red squares.

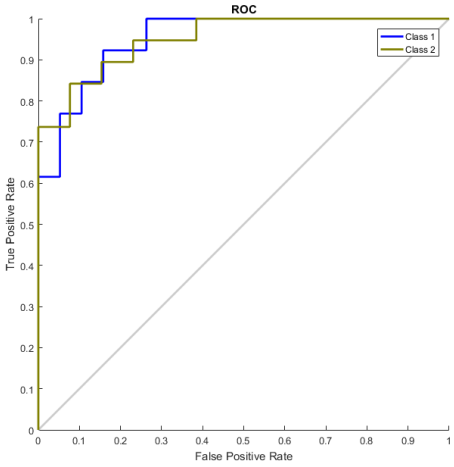


Sample confusion matrix for testing dataset with three target classes.

```
Matlab:-> [net,tr] = train(net,x,t);
testX = x(:,tr.testInd); testT = t(:,tr.testInd);
testY = net(testX); plotconfusion(testT,testY)
```

		Predicted Condition		
		Predicted +ve	Predicted −ve	
True Condition	Condition +ve	True⁺	False⁻ (Type-II error)	**Sensitivity or Recall** = $\frac{\sum True^+}{\sum Condition^+}$
	Condition −ve	False⁺ (Type-I error)	True⁻	**Specificity** = $\frac{\sum True^-}{\sum Condition^-}$
	Prevalence = $\frac{\sum Condition^+}{\sum Total}$	**Precision** = $\frac{\sum True^+}{\sum Outcome^+}$	**Negative Predictive Value** (NPV) = $\frac{\sum True^-}{\sum Outcome^-}$	**Accuracy** = $\frac{\sum True^+ + \sum True^-}{\sum Total}$

- Another measure of how well the neural network has fit data is the "receiver operating characteristic" (ROC) plot. It shows how false +ve and true +ve rates change as "threshold" for output classification is varied from 0 to 1. This is straightforward for binary classification. Ideally, we are looking for the curve to bow towards the top left corner. A completely random model with no utility will yield an ROC plot that is diagonal at 45%. One should pick a threshold that best balances the two rates to meet the needs of the application. For multi-class problems, each class will be compared against all other classes, yielding one ROC curve for each class (not quite effective).



Sample ROC curve plot for a binary classification problem (cancer detection). (Class 1 indicates cancer patients and class 2 normal patients)

```
Matlab:-> plotroc(testT,testY)
```

- Class Imbalance: Comes into play when observations are not uniformly distributed across the different classes (e.g., outcome of "strep test" for patients with sore throat condition is mostly "negative" for Streptococcal pharyngitis, also known as strep throat).
  - For binary classification problems with class imbalance, one option is to use the "balanced" F-Score (F_1 Score), which effectively integrates both precision and recall:

$$F_1\ Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

  - F_1 Score vs. Accuracy: Suppose we are dealing with weather prediction where only 10% of the days are "sunny". A model that (almost) always predicts "not sunny" (say 99.999% of the time, randomly) has an accuracy of 90% whereas it's F_1 score will be 0 (Precision will be ≈ 0/(0+0) NaN and Recall will be 0/(0+10)=0).

Model Always Predicting Not-Sunny (randomly, 99.9% of the time):

			Predicted Condition			
			P(Sunny)	P(Not)	Accuracy:	89.92%
		Cases	0.001	0.999	Recall:	0.001
True Condition	Sunny	10	0.01	9.99	Precision:	0.1
	Not	90	0.09	89.91	F1 Score:	0.00198

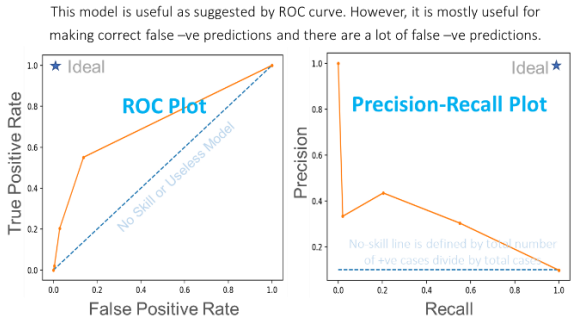
Perfect:

			Predicted Condition		Accuracy:	100.00%
		Cases	P(Sunny)	P(Not)	Recall:	1
True Condition	Sunny	10	10	0	Precision:	1
	Not	90	0	90	F1 Score:	1

"Decent":

			Predicted Condition		Accuracy:	97.00%
		Cases	P(Sunny)	P(Not)	Recall:	0.9
True Condition	Sunny	10	9	1	Precision:	0.818182
	Not	90	2	88	F1 Score:	0.857143

- ROC plot is not that effective under class imbalance. Precision-Recall curve is more useful. It does not make use of true negatives and is only concerned with correct prediction of minority class. For more discussion, check out this link.



ROC and Precision-Recall Curves for a Weak Model Learnt from a Highly Imbalanced Dataset (Source: Link)

- Cost Sensitive Learning: In cases where the cost of false positives is different from cost of false negatives, cost sensitive learning will be necessary. For example, one can pick a classifier based on a threshold point on the ROC that minimizes the overall cost.

TIME-SERIES FORECASTING

- Partition the time-series as consecutive segments for training, validation, and testing (with no overlaps) to demonstrate generalization ability (e.g., $1{:}\,n$ observations for training, $n+1{:}\,n+m$ observations for validation, and $n+m+1{:}\,n+m+p$ observations for testing).
- MSE plots during training are meaningful for time-series forecasting as well.
- Some popular measures for evaluating time-series predictions are:
  - *MAPE (Mean Absolute Percent Error):* Measures the size of the error in percentage terms. It is calculated as the average of the unsigned percentage error.

$$\left(\frac{1}{n}\sum\frac{|Actual - Forecast|}{|Actual|}\right)*100$$

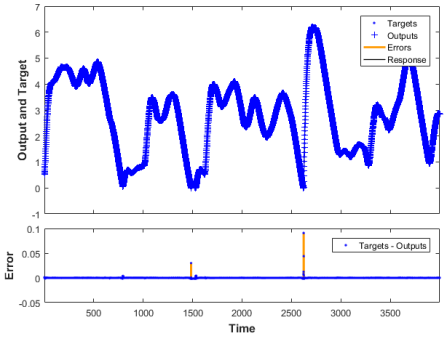
Month	Actual	Forecast	Absolute Percent Error
1	112.3	124.7	11.0%
2	108.4	103.7	4.3%
3	148.9	116.6	21.7%
4	117.4	78.5	33.1%
MAPE			**17.6%**

- MAPE is scale sensitive and should not be used when working with low-volume (scale) data. Notice that because "Actual" is in the denominator of the equation, MAPE is undefined when Actual value is zero. Furthermore, when the Actual value is not zero, but quite small, the MAPE will often take on extreme values. This scale sensitivity renders the MAPE close to worthless as an error measure for low-volume data.
- *MAD (Mean Absolute Deviation):* Measures the size of the error in units. It is calculated as the average of the unsigned errors, as shown in the example below:

$$\frac{1}{n}\sum|Actual - Forecast|$$

Month	Actual	Forecast	Absolute Error
1	112.3	124.7	12.4
2	108.4	103.7	4.7
3	148.9	116.6	32.3
4	117.4	78.5	38.9
MAD			**22.08**

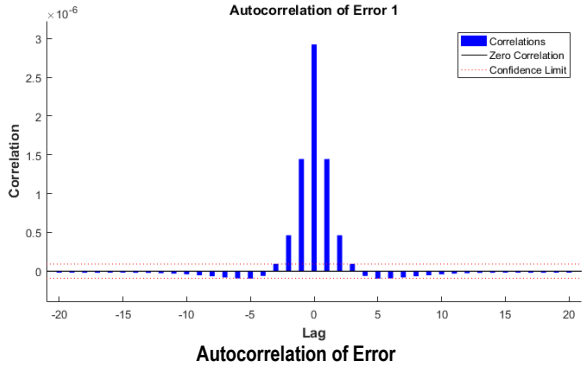
- Plot actual vs predicted values for the testing times-series segment along with the errors.



Sample forecasting plot.

```
Matlab:-> plotresponse(Ts,Y)
```

- Plot correlation of error at time t, $e(t)$ with errors over varying lags, $e(t + lag)$. The center line shows the mean squared error. If the network has been trained well all the other lines will be much shorter, and most if not all will fall within the red confidence limits.



Autocorrelation of Error

```
Matlab:-> E = gsubtract(Ts,Y); ploterrcorr(E)
```

- Model should outperform a naïve model which predicts $\hat{Y}(n + 1) = Y(n)$
- Test accuracy of recursive predictions if relevant.