**\* 실행 환경** : MATLAB R2017a

“스크립트 내 함수” 기능을 사용하는데 R2016b부터 지원하는 것으로 알고 있습니다.

**\* 변수 사용**

Continuous, discrete, ordinal, nominal 모든 변수들을 사용하였습니다. Ordinal 변수는 연속적으로 증가하는 정수값들을 할당하였고 Nominal 변수는 해당 변수의 category 종류만큼의 Boolean 변수들로 분해하였습니다.(해당 기능을 포함한 data import관련은 AMES.m 파일에 구현되어 있습니다.)

**ExterQual (Ordinal)**

Rawdata: Po, Fa, TA, Gd, Ex  
Processed: 0, 1, 2, 3, 4

**MSSubClass (Nominal)**

Raw Data: 20, 30, 40, 45….  
Processed: MSSubClass\_20 - 0 or 1  
 MSSubClass\_30 - 0 or 1  
 …

학습 효율을 높이기 위해 확장된 변수들을 실제 사용하기 전, [-1, 1] 혹은 [0, 1]로 Standardization 과정을 거쳤습니다.

**\* Performance Check w/o additional test data**

추가적인 테스트 데이터 없이 모델의 성능을 측정하기 위해 모든 모델에 K-Fold Cross Validation을 적용하여 Training Set과 Test Set이 겹치지 않게 하였습니다.

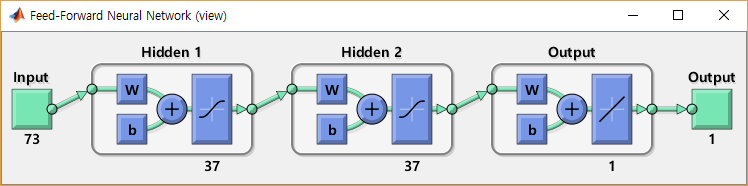
**\* KNN & MLP**

모든 문제는 KNN과 MLP로

**A. [Price prediction]**

**1.KNN**

**1.MLP**



Hidden Layer를 2 개로 하였으며 각 Hidden Layer의 노드는 Dim(Input) 반으로 하였습니다. regression문제라 output node의 activation function을 linear로 하였으며 Hidden Layer의 activation function은 sigmoid로 하였습니다.

Loss Function : Mean Square Error(MSE)

**- 2000 epochs 실행과 MSE -**



MSE 를 LossFunction으로 하여 epoch에 따라 MSE가 감소하는 것을 볼 수 있지만 learning rate조절 같은 별다른 처리 없이 기본적인 MLP로 구현되어 있어 학습속도가 꽤 느린 것을 볼 수 있습니다. 또한 overfitting으로 인해 TestMSE의 감소속도가 상대적으로 느린 현상도 나타납니다.

**- 실행 결과 표 -**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Training | | | | Test | | | |
| epochs | Bias | MD | MAD | MSE | Bias | MD | MAD | MSE |
| 1 | 11.7 | 504471.4 | 34503.4 | **2472711663.1** | -397.2 | 279549.5 | 37510.8 | **2758683336.1** |
| 500 | -10.6 | 501244.7 | 34034.9 | **2417893138.8** | -440.8 | 281271.5 | 37395.8 | **2754422249.3** |
| 1000 | -8.2 | 497610.2 | 33659.5 | **2372882426.9** | -448.5 | 281430.0 | 37222.2 | **2744701549.0** |
| 1500 | -7.0 | 494341.5 | 33324.9 | **2332988745.7** | -438.2 | 280972.9 | 37037.4 | **2732209940.4** |
| 2000 | -6.3 | 491417.9 | 33020.8 | **2296892472.2** | -420.1 | 280255.1 | 36852.2 | **2718982451.9** |

LossFunction인 MSE는 꾸준히 감소하는 경향을 보이고 Test Set의 Bias, MD의 경우 중간중간 값의 증가도 보이고 있습니다.

**추가 데이터로 실행하는 방법**

regression\_ffn.m 의 useAdditionalTestData를 true로 설정하고 additionalTestDataFile을 원하는 파일 이름으로 설정합니다(경로 포함).

useAdditionalTestData = true;

additionalTestDataFile = '../../data/ml\_project\_test.csv';

**2.MLP**

PCA를 활용하여 중요한 변수와 그렇지 않은 변수를 구분하였습니다.(paramReduction.m) PCA를 수행하여 기존 변수의 Linear Combination형태의 새로운 변수를 사용할 수도 있지만 직관적이지 않다고 생각하여 약간 다른 방법을 사용하였습니다. Covariance matrix의 eigenvalue크기 순서대로 정렬하였을 때 아래쪽에 있는 PC들은 많은 정보들을 포함하고 있지 않은 중요하지 않은 축이라고 볼 수 있고, 중요하지 않은 축들에 대부분의 coefficient들을 포함하고 있는 변수들은 다른 변수들에 비해 중요하지 않다고 볼 수 있습니다. 이런 방법으로 중요한 변수 73개를 따로 분리하여 사용하였습니다.

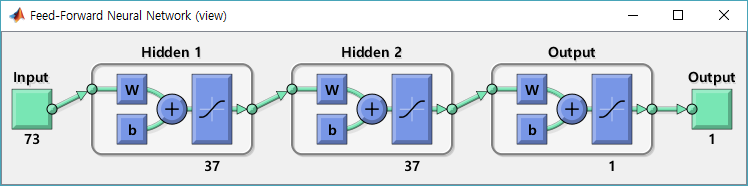
paramReduction.m을 실행하면 중요한 PC 31개를 제외한 중요하지 않은 PC들에 squared sum 1 중에 0.85이상을 포함한 73개 변수들의 index들을 출력합니다.

실행하면 결과가 perf가 크게 다르진 않고 실행 속도가 더 빠르다는 얘기

**B. [Price range classification]**

**1.KNN**

**1.MLP**



regression문제와 같이 Hidden Layer를 2 개로 하였으며 각 Hidden Layer의 노드는 Dim(Input) 반으로 하였습니다. classification문제라 output layer를 포함한 모든 activation function을 sigmoid로 하였습니다

- In-Sample Error Rate (1260 training, 1260 test) -

Training Set과 Test Set이 같으므로 Training Error와 Test Error가 같음

- 5-Fold Error Rate (1008 training, 252 test) -

**2.KNN**

**2.MLP**

A.2.MLP와 동일한 방법으로 73개의 중요 변수들을 추출하여 사용하였습니다.

**3.KNN**

**3.MLP**

주어진 Sample 1260개를 S1, S2, …, S6으로 6등분한 후 S1~S5로 5-Fold Cross-Validation을 사용하여 5개의 MLP모델을 학습합니다. 미리 학습된 5개의 MLP모델의 Majority Vote방식으로, 학습에 사용되지 않은 S6을 test data로 하여 MLP-Ensemble모델의 성능을 측정하였습니다.

**- classification\_ffn\_ensemble.m의 실행 결과 -**

ModelNo: 1, TestPerf = 0.11905  
ModelNo: 2, TestPerf = 0.15238  
ModelNo: 3, TestPerf = 0.14286  
ModelNo: 4, TestPerf = 0.14762  
ModelNo: 5, TestPerf = 0.1619  
**EnsemblePerf = 0.13333**

MLP-Ensemble 모델이 0.13333의 ErrorRate을 보여주었으며 Majority Vote에 사용된 Model1보다는 안 좋은 결과를 보여주었지만 다른 4모델보다는 나은 결과를 보여주었습니다.

**추가 데이터로 실행하는 방법**

classification\_ffn\_ensemble.m의 testDataFile을 원하는 파일 이름으로 설정합니다(경로 포함).

testDataFile = '../../data/ml\_project\_test\_ensemble.csv';