CS376 Term Project

Team 22

December 13, 2018

1. Model Descriptions

1.1 Data Preprocessing

18번 col인 construction completion date는 19번 col에 이미 built year정보가 있고, 값이 없는 부분도 있어 18번 col을 제외해도 괜찮다고 판단했다. 그러하여 input feature는 총 22개이다.

Training에 들어가기 전에, 0번 col이 str 타입이기 때문에, 월과 일을 버리고 년도만 따와 float 타입으로 변환시켜주었다.

1, 2, 3, 8, 11, 12, 15, 17, 19, 21번 col의 빈칸은 모두 해당 col의 평균값으로 채운 뒤, scikit-learn 에서 제공하는 standard scaler를 사용하여 normalization 한다.

$$z = \frac{x - \mu}{\sigma}$$

$$\mu = \text{Mean}$$

$$\sigma = \text{Standard Deviation}$$

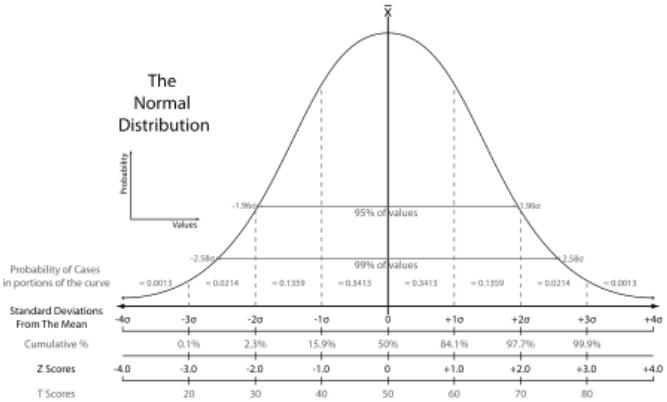

Figure 1. standard scale normalize equation and distribution

Price col은 다른 input feature들과는 다르게, 일단 한번 log를 씌운 뒤에 normalization 한다. 이

렇게 한 이유는 우리의 평가 기준이 'label과 output 사이의 오차율'이기 때문이다. Log를 씌우지 않는다면 작은 label값의 오차와 큰 label값의 오차가 동일한 비중을 갖게 되어, 작은 label을 가지는 data에 대해서 상대적으로 큰 오차율을 허용하는 방향으로 학습될 것을 우려했다. 실제로 log를 씌우는 과정을 빼면, 약 1%정도 오차율이 늘어나는 것을 볼 수 있었다.

1.2 Neural Net

우리 모델은fully connected layer로만 이루어진 neural net이다. 각 layer의 size는 (22, 100), (100, 100), (100, 100), (100, 1) 이렇게 총 4개로 정했고, 다음 layer로 들어가기 전에 activation function으로는 모두 ReLU를 사용했다. Batch size는 256으로 정했다. 아래는 우리 모델의 대략적인 모습과, 우리 모델이 input으로부터 output을 도출해내는 과정을 수식으로 나타낸 것이다.

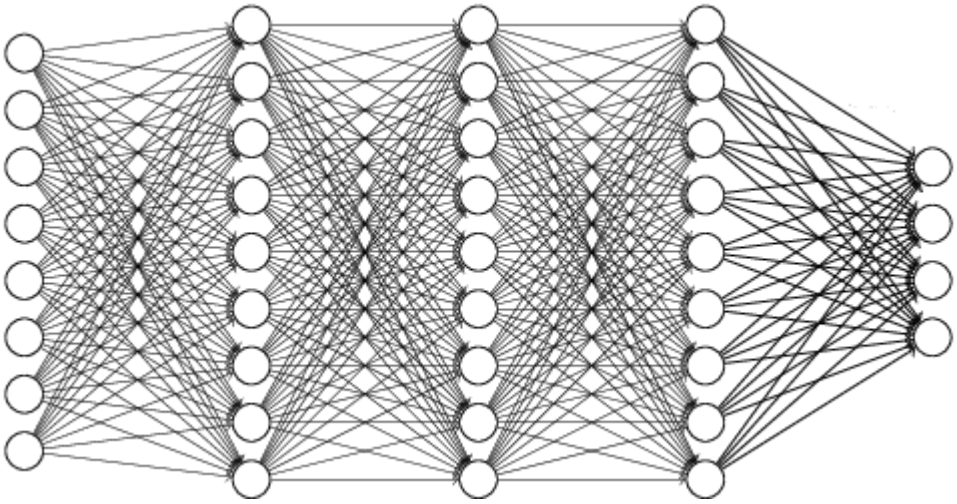

Figure 2. 4개의 fully connected layers. Neuron의 개수는 우리 모델과 조금 다르다.

$$\text{input} = (\text{num of data}, 22) \text{ matrix}$$

$$\text{fc1} = (22, 100) \text{ matrix}$$

$$\text{fc2} = (100, 100) \text{ matrix}$$

$$\text{fc2} = (100, 100) \text{ matrix}$$

$$\text{fc4} = (100, 1) \text{ matrix}$$

$$\text{output} = (\text{num of data}, 1) \text{ matrix}$$

$$\text{output} = \text{relu}(\text{relu}(\text{relu}(\text{input} * \text{fc1}) * \text{fc2}) * \text{fc3}) * \text{fc4}$$

Training 은 20 epochs를 반복했다. Optimizer는 Adam을 사용했고, loss는 MSE(Mean Squared Error, = L2) loss를, learning rate는 0.01에서 시작하여, 다음 epoch로 넘어갈 때마다 1.3배씩 감소하게 하여 최대한 local minimum에 가까이 갈 수 있도록 해주었다.

2. Unique Methods

2.1 Gaussian Normalization

앞서 1.1에서 언급했듯이 우리는 data preprocessing 단계에서 (빈 값은 해당 column의 평균을 넣어주고) Gaussian Normalize를 했다.

	No Normalization	Gaussian Normalize	MinMax Normalize
오차율	30%	5.4%	6.1%

표 1. Normalize 하지 않은 오차율, Gaussian Normalize 오차율, MinMax Normalize 오차율

우선 Normalization의 유무에 따른 결과의 차이는 상당했다. Gaussian Normalization을 했을 경우, Price 오차율은 5.4%였다(data_train.csv). Normalization을 하지 않았을 경우의 오차율은 30%였다(data_train.csv). 약 25% 정도의 차이가 발생했는데, 이러한 차이는 각 Column 마다 범위가 달라, Neural Net에서 Price를 예측하는 각 column 별 weight와 bias를 찾기 어려워져서 발생했다고 추정한다. 예를 들어, 22번 Column의 경우 The number of subway stations near the apartment인데, 이 값의 최대값은 2이고, 최소값은 0이다. 12번 Column의 경우 The total area of parking lot인데, 이 값의 최대값은 365115이고, 최소값은 15340이다. Neural Net은 학습을 하며 각 Column 별 가중치를 만들지만 각 column의 range가 174,887배 차이가 남으로 이에 따른 가중치 계산에 많은 학습 시간을 할당할 것이고, local minimum에 빠질 확률도 더 커질 것이다.

그러므로 각 Column의 중요도를 모르는 우리는 Data를 Normalize 해줌으로써 Neural Net이 좀 더 학습이 용이하게 만들어 주었다.

그 후, Gaussian Normalize와 MinMax Normalize의 결과를 비교하였다. MinMax Normalize는 Min 값과 Max 값을 이용하여 Scale(우리는 0 ~ 1의 범위를 이용) 범위 내로 Normalize 해주는 것이다. Gaussian Normalization을 했을 경우 오차율은 5.4%였고, MinMax Normalization의 경우 6.1%의 오차율이었다. 그래서 우리는 Gaussian Normalization을 사용했다.

2.2 Adam Optimization

앞서 1.2에서 언급했듯이 우리는 Adam을 optimizer로 사용했다. Optimizer의 종류에 따라 Neural Net에서의 학습 방향이 달라진다. 따라서 우리는 optimizer를 Adadelta, Adagrad, Adam, Adamax, ASGD, RMSprop, Rprop, SGD로 바꿔가며 결과를 확인했다. 결과는 아래 표 2와 같다.

	Adadelta	Adagrad	Adam	Adamax	ASGD	RMSprop	Rprop	SGD
오차율	17.0%	6.5%	5.3%	5.5%	14.6%	6.3%	10%	14.5%

표 2. 각 optimizer 별 오차율

결과에서도 보이듯 Adam의 오차율이 제일 적었다. 그리하여 우리는 Adam을 우리의 Optimizer로 선택했다. Adam이 다른 Optimizer의 결과와 비교했을 때 좋은 이유를 생각해보면, Adam이 갖는

특유의 장점에서 비롯되었다고 생각한다. Adam의 특징은 다음과 같다. Adam은 Adagrad와 RMSProp의 장점을 합친 방법이다. Adam의 경우 stepsize가 gradient의 rescaling에 영향 받지 않고, gradient가 커져도 stepsize는 boud되어 있기에 어떠한 function을 사용하더라도 안정적으로 최적화를 위한 하강이 가능하다는 장점이 있다.

3. Libraries

3.1 python

Version: 3.5.2

Purpose: 프로젝트에 주어진 언어

3.2 torch

Version: 0.4.1

Purpose: neural net을 사용하기 위해

3.3 torchvision

Version: 0.2.1

Purpose: torch.utils.data의 DataLoader와 Dataset을 사용하기 위해

3.4 numpy

Version: 1.15.4

Purpose: 행렬 표현과 행렬 계산을 위해

3.5 pandas

Version: 0.23.4

Purpose: csv 읽기, 쓰기와 데이터 preprocessing을 위해

3.6 scikit-learn

Version: 0.20.1

Purpose: 데이터 preprocessing(normalization)을 위해

3.7 pickle

Version: 4.0

Purpose: train model 저장을 위해

4. Source codes

Team22의 파일 구조는 다음과 같다.

CS376

|−−−> README // Readme file

|−−−> data_test.csv //csv file containing datas for test

|−−−> data_train.csv //csv file containing datas for train

|−−−> team22_train.py // main python file for training

|−−−> team22_test.py // main python file for testing

|−−−> data_train_processed.csv // processed csv file from data_train.csv

|−−−> unique.csv // csv file containing predicted values of apartment prices from data_test.csv, using unique method

|−−−>base.csv // csv file containing predicted values of apartment prices from data_test.csv using base method

|−−−>team22_nn_[mode].sav // saved model. read README about mode

|−−−>input_scaler_[mode].sav // saved input scaler

|−−−>price_scaler_[mode].sav // saved price scaler

|−−−>means_[mode].sav // saved column mean

|−−−>predict_[mode].sav // saved predicted result(unique.csv = predict_1.csv, base.csv = predict_4.csv

4.1 Correspond to the model components

```
# neural net
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(num_inputs,100)
        self.fc2 = nn.Linear(100,100)
        self.fc3 = nn.Linear(100,100)
        self.fc4 = nn.Linear(100,1)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = self.fc4(x)
        return x
```

team22_train.py의 82~95번 줄에 있다.

team22_test.py의 81~94번 줄에 있다.

4.2 How to implement another test set.

다른 test file을 적용하려면 team22_test.py의 40번줄을 변경하면 된다.

training을 위해 "python3 team22_train.py"를 실행한다.

Console 창에 1, 2, 3, 4 입력한다. 각 번호의 의미는 다음과 같다.

1 : Team22의 Unique Method인 normalization, Adam optimizer를 사용한다. Unique Method

2 : Normalization을 하지 않고 Adam optimizer만 사용한다

3 : Normalization을 하고 Adam 대신 Adagrad optimizer를 사용한다.

4 : Normalization을 하지 않고, Adam 대신 Adagrad optimizer를 사용한다. Base Method

test를 위해 "python3 team22_test.py"를 실행한다.

4.3 Hyper-parameters to use for test

4.3.1 Adam optimizer

Learning rate: Base 0.01, Divide it by 1.3 in every epoch.

Betas: 0.9, 0.99

eps = 1e-8

weight_decay(L2 penalty) = 0

amsgrad = False

4.3.2 Neural Net Model

The number of layers : 4

Layer size : (22, 100), (100, 100), (100, 100), (100, 1)

Batch size : 256

Activation function : ReLU

Epoch : 20

5. Performance

5.1 About splitting the samples in train data for training and validation

np.random.uniform()을 사용하여, data_tran.csv의 데이터를 9대 1의 비율로 training set과 validation set으로 나누어 모델을 만들어 테스트하였다. np.random.uniform()을 사용한 이유는 origin data를 보니 어느 정도 경향성이 비슷한 데이터끼리 인접해 있다고 판단하여, Overfitting을 막기위해 랜덤하게 선택하였다.

5.2 Performances on the training and validation samples

Validation data로 |{(실제 값) - (예측 값)} / (실제 값)| 을 한 결과, 5.3 %의 오차가 발생하였음을 알 수 있었다.

5.3 Execution time on test data

mode 값을 input으로 받은 후부터 preprocessing과 만들어 놓은 Model를 이용하여 predict하는 시간을 포함한 program 종료까지 걸리는 시간은 2.9 sec 이다.

5.4 Analysis of the performance

Figure 3은 X축을 실제 값, Y축을 예측 값으로하여 plotting을 해본 결과이다. 그림 상에서 알 수 있듯이, 전반적으로 Y=X의 그래프를 따르며, 오차율이 낮음을 확인 할 수 있었다.

Figure 4는 각 data의 에러의 백분율 분포를 히스토그램으로 나타낸 것이다. 전반적으로

Gaussian graph 형태를 띄고 있는 것을 볼 수 있었다. Data에 gaussian noise가 존재하며, 평균이 0에 가까운것으로 보아 Bias가 작음을 알 수 있다.

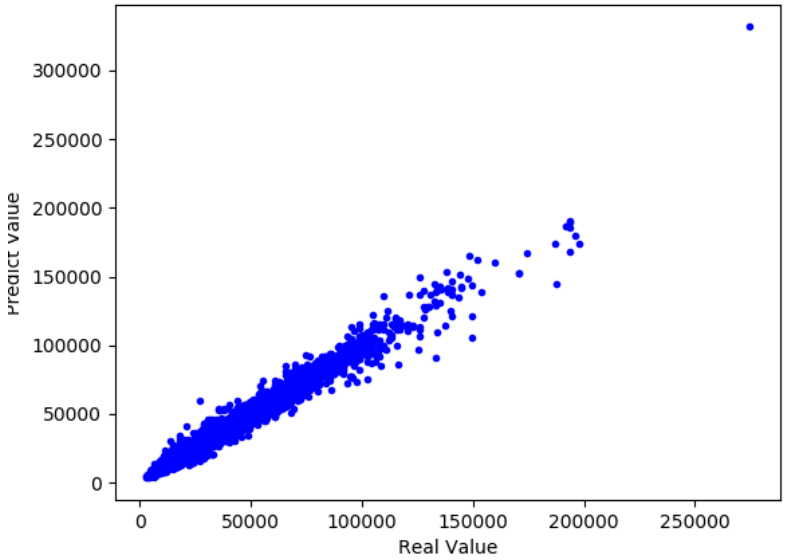

Figure 3. X : 실제 Price, Y : 예측 Price (data_train.csv 이용)

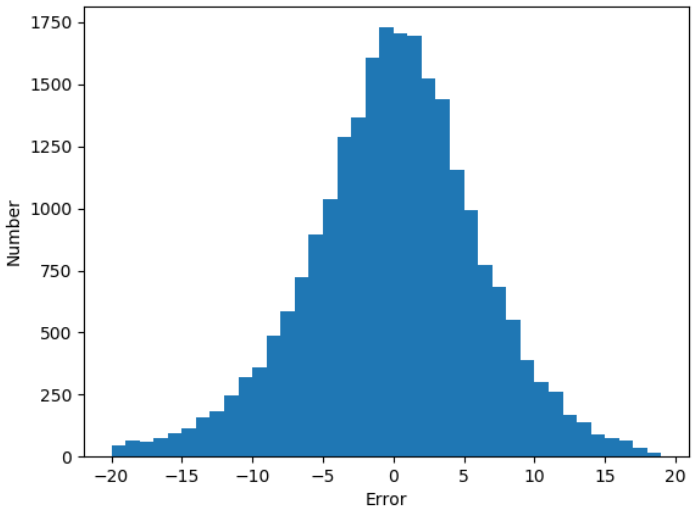

Figure 4. 각 data의 에러 백분율 히스토그램