**ECS 271 Final Report - House Price Prediction**

Team members: Kai Lan, Xiang Hao

Professor Name: Hamed Pirsiavash

Subject Name: ECS 271 Machine Learning & Discovery

- Missing data

- Correlation between features

- Sparse tensor

- Decaying learning rate

- User defined weights for different features

Unlike the homework data that are ready to go, our project data requires preprocessing. The data come in both numeric and non-numeric forms, and missing data are also present.

Avoiding overfitting: dropout, data augmentation, ensemble different models.

- batch normalization:

learm two paramters

- weight normalization:

- Very cool but not implemented on MPS

- learn two parameters: weight magnitude and weight direction

- correct features ordering: https://medium.com/spikelab/convolutional-neural-networks-on-tabular-datasets-part-1-4abdd67795b6

- Tabular data augmentation

- Few-Shot learning

- Kernel 2D CNN

FNN 测试结果显示，三层fl效果好于四层，size：305-64， 64-32， 32-1

FNN + dropout: bad effect when adding dropout, the loss punctuates much and is generally higher than without dropout.

Outline:

* The problem that we are solving – Background
* Motivation: why is this an important problem to solve – Problem
* What method we used, and problems we encountered – Methodology
* What dataset we used
* Our results including comparison with some baselines (that are simpler methods). The baseline can be our method excluding an important component. Ideally, we want to show that our method works better than the baselines – Result

**Background**

Currently, the world economy is unstable. In this case, the situation of real estate is continuously fluctuating as the world economy fluctuates. It is important for us to provide the user a relatively reliable system to estimate the price of their dream house since sometimes, the user has no ideas about how they can estimate house prices under current real estate market especially when the house’s price is always changing. Therefore, a system with a comprehensive price evaluation function can be very helpful to some customers who have a need to buy a house. Once they get to know the market price of the house, they will not be easily misled by some deceiving house agent.

A potential home buyer would be interested in a good estimate on the price of their dream house so that they can plan their budget and make decisions more advisedly. There are many factors that come into play for a house quote, and not all of them are straightforward for those who are new to the real estate market.

Our study here provides 79 variables to estimate the market value of a house of given types to comprehensively estimate the house price in the current market and help those buyers eliminate concerns about real estate bubble and incomplete housing price estimates.

Our study is based on data from past house sales on the market in Ames, Iowa, and is aimed to find a correlation between the sales price and the information related to those 79 variables. Although this dataset is restricted to one area, we can easily generalize our model to data in other regions as well in the future.

**Problem**

For better predictions on housing prices, we hoped to explore machine learning techniques to study the implicit relations between characteristics of a house and its market value. This comes down to a regression problem. A good model is the one whose predicted value is close to its label value, as well as generalizing it to other houses it has not seen before. Ideally, the smaller the error is between predictions and ground truth, the better the model is. Mathematically, we would compute the losses for both training and test datasets, and hopefully both of them go down as we run more training cycles. By computing the gradient of the loss function, we can update the parameters for our model using gradient descent.

**Methodology**

* *Data processing*

Our dataset can be downloaded from Kaggle (<https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/data>). The data set presents a large amount of missing data for each feature, ie, NaN values. We had to validate those values in some way before using them for training. To fill in the missing values, we split the entire dataset into categorical features and numeric features, for the former we treat NaNs as a special category, and for the latter we replace them with their corresponding average values. Then, we had to convert all data into numeric for training purposes. One-hot encoding is a reliable way of doing this, so we adopted it. Furthermore, all the features values and their labels are normalized with their means and standard deviations. For visualizing our datasets, please refer to Jupyter Notebook data\_analyzer.ipynb.

* *Linear regression*

Linear regression is an old and simple approach in machine learning. It assumes that dependent variables are linear to independent variables. To achieve this, we can simply use a neural network with one fully connected layer that takes all the features as input and outputs a single value as the predicted price without activation. This is the basic model we will compare other methods to later.

* *One-dimensional Convolutional Neural Network*

Our architecture was inspired by

(<https://medium.com/spikelab/convolutional-neural-networks-on-tabular-datasets-part-1-4abdd67795b6>). This architecture was claimed to be able to sort one-dimensional tabular data to exploit the correlation between.



As seen from the above figure, the input vector is first extended a longer vector before it is passed into the convolution layers. After all the convolutional layers, the data are flattened intn a long vector and is passed through the output layer. Then the loss and its gradient are computed and all the weights are updated. Batch normalizations were used to more neural network faster and more stable (https://arxiv.org/abs/1502.03167). Originally we thought about using weight normalization, since it was proven to improve the training (<https://arxiv.org/pdf/1602.07868.pdf>). However, in Pytorch it has not yet supported MPS backend on Mac M1 machines.

* *Two-dimensional Convolutional Neural Network*

In order to transform our one-dimensional data into two-dimensional, we applied outer product to each input vector before convolution operator. The rest of this method is very similar to 1D CNN. Then the data traverse three convolutional layers,



**Results**

Now we compare our 1D and 2D CNN models with the baseline approach. To evaluate the models, we compute the relative error From observation, there are some predictions dramatically deviated from its label value, ie, greater than , which we consider outliers. Model 1 and 2 have less outlier predictions.

[Picture show histogram]

**Future improvement**

There are many improvements we can make if we had more time. One problem is that our dataset is too small, with only 1460 rows and 81 columns. We hope to study effective training methods for small datasets, such as one-shot training or few-shots training.

What method we used, and problems we encountered

* Method we used:
  + Convolution Neural Network: In the CNN class, we used Linear, Conv2d, Maxpooling, ReLU, Dropout, and BatchNorm1d.
  + Kernel Method: We tried to add Kernelized Perceptron method into our model,
  + Tabular data augmentation: Used it to process data.
  + Few-Shot learning
  + Kernel 2D CNN
  + Batch Normalization: We used it in CNN.
* Problems we encountered:
  + Correlation between features: The correlation among features is not closed or is independent that is totally different from homework.
  + Sparse tensor:
  + Decaying learning rate:
  + User defined weights for different features
  + Unlike the homework data that are ready to go, our project data requires preprocessing. The data come in both numeric and non-numeric forms, and missing data are also present.
  + Overfitting: We used dropout, data augmentation, ensemble different models to avoid overfitting.

What dataset we used

Our results including comparison with some baselines (that are simpler methods). The baseline can be our method excluding an important component. Ideally, we want to show that our method works better than the baselines.