# NTU ML 2017 Final

## Sberbank Russian Housing Market　俄羅斯房地產

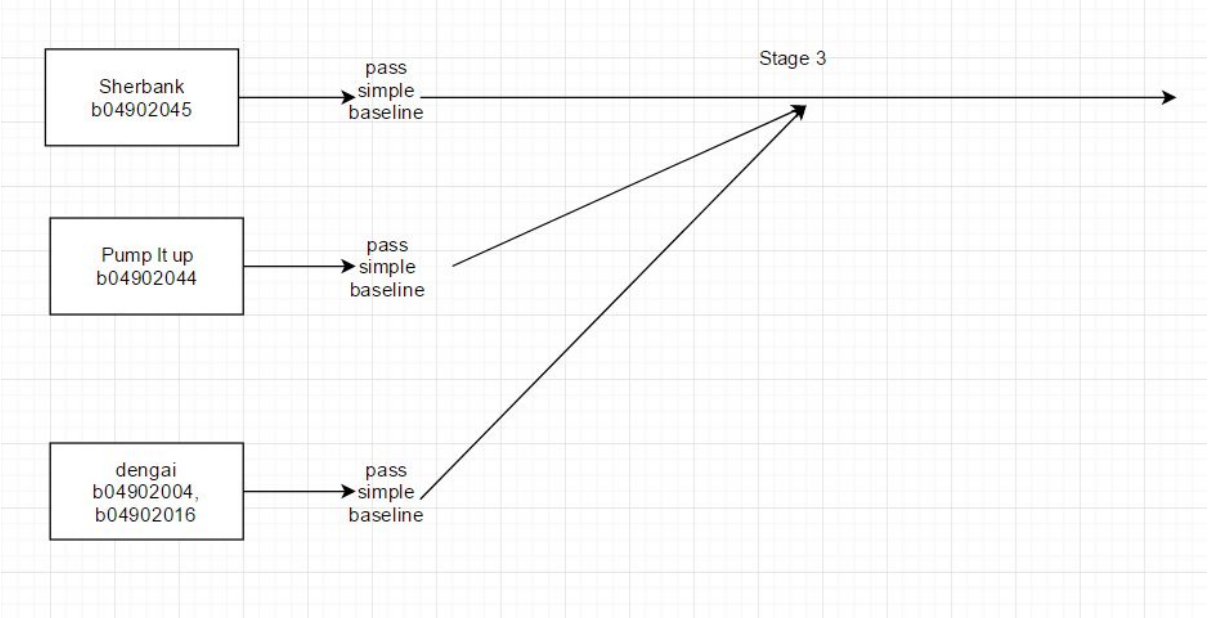
**Team name: 大玉螺旋丸**

**Members:**

王佑安 b04902004 曾奕青 b04902016 朱柏澄 b04902044 孫凡耘 b04902045

**Work division：**

我們的分工是剛開始分配到不同題目，最後決定一起做這題。




## Gradient Boosting Regressor

1. Introduction

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

除了看到許多kernel外，開始這個比賽前就先去查了以往kaggle上的regression的比賽的winner's solution. 發現最少有一半以上有gradient boosting，更確切一點應該是ensemble的model中幾乎少不了gradient boosting的model，也不乏使用一推gradient

boosting model再做ensemble的。套件使用xgboost。

## 2. Preprocessing/Feature engineering

這個dataset第一個觀察就是很多缺少的值，而且不只很不完整，甚至有不少錯誤(錯誤是指不合常理的數值，例如生活坪數大於總坪數。發生這樣這樣的情況我們以nan取代)。這時候有兩個選擇，我可以drop掉那些不合理或缺少值的data或選擇做一些調整。

Attempted appoarch to handle NaN：

- Replace with mean
- Replace with kinetic energy(also called information energy)

Reference: https://alexandrudaia.quora.com/Kinetic-things-by-Daia-Alexandru

- Let the model handle it (without explicitly handling it)
    - Might replace it with 0
    - automatically "learn" what is the best imputation value for missing values based on reduction on training loss.

Observation of abnormal data

- life_sq >= full_sq
- life_sq < 5 or full_sq < 5
- build_year can be in future

再來就是我們有嘗試對training data用一個特殊的方法做under sampling，原因我們會再discussion中討論。

New features

- week/month/year 交易筆數
- 經緯度資料(有人在discussion中分享)
- floor / max_floor
- kitch_sq / full_sq
- life_sq / num_room

Features removed(根據常理判斷與xgboost的feature importance作為依據)

附圖為我們code的截圖:

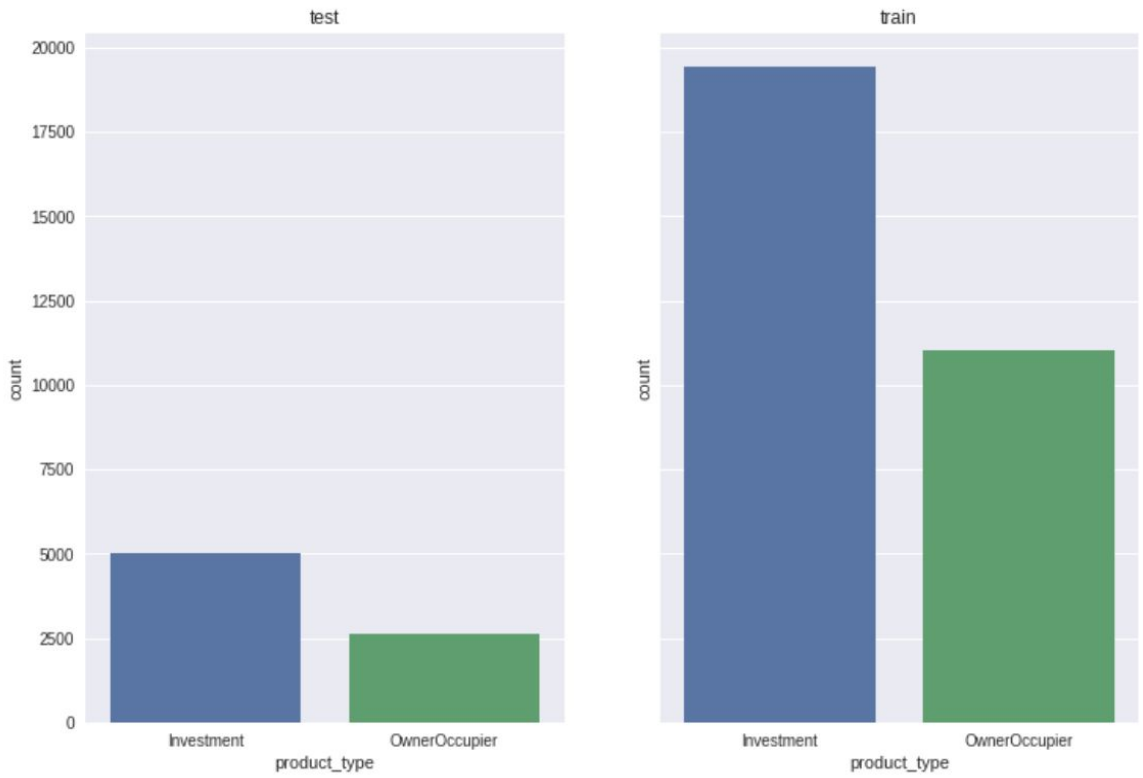
```
useless = [ "ID_railroad_station_walk", "ID_big_road1",
        "ID_railroad_terminal", "ID_metro", "ID_railroad_station_avto",
        "ID_big_road2", "ID_bus_terminal" ]
x_train = x_train.drop( useless, axis=1 )
x_test = x_test.drop( useless, axis=1 )
```

## 3. Experiments and Discussion

相信開始這個比賽的時候，很快的就會觀察就cross validation怎麼樣都做不準，傳上去比自己的validation差很多。我們嘗試了很多validation的方法。除了一般的validation以外，我們也嘗試validate再最後的data上，對後predict的時候再全部下去train，這樣嘗試的原因就是training跟testing data set有一個時間先後的關係，所以做validation的時候也試圖模擬我們predict testing data set時的這種時間先後的關係。不過這樣的validation似乎並不好，所以我們最後採用的是Method 2的validation方法，不過從下面的眾model中可以看出，這樣的validation顯然還是有相當大的問題，只有在xgboost上validate似乎比較正常。

Validation(adjust distribution會在下面解釋)
Method 1: random train_test_split
**Method 2: adjust distribution + random train_test_split**
Method 4: adjust distribution + chose validation set on last 4000
Method 4: adjust distribution + chose validation set on last 5000

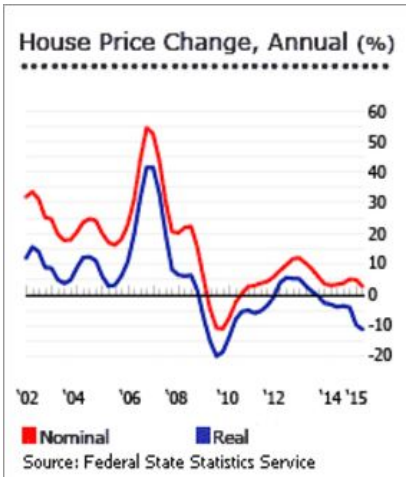
第一個想法就是：因為我們預測的年份是2015~2016，但training data是從2011~2015。或許是因為這兩個時期的大環境因素差很多？所以就把macro feature也加進去training and validation，但仍然沒有改進。後來才發現，做training 跟testing data的有些feature有很不一樣的distribution。




這張圖可以看到testing set裡面OwnerOccupier這種product_type(超過Investment的一半)比在training set中OwnerOccupier佔得比例大得多。所以後來把training data做適當的under sample, validation的情況便好很多，雖然仍然無法到非常接近leaderboard的prediction，但大致上趨勢可以對。

```
train = train.drop(train[(train.price_doc <= 1e6) & (train.product_type == 'Investment' )].index)
```

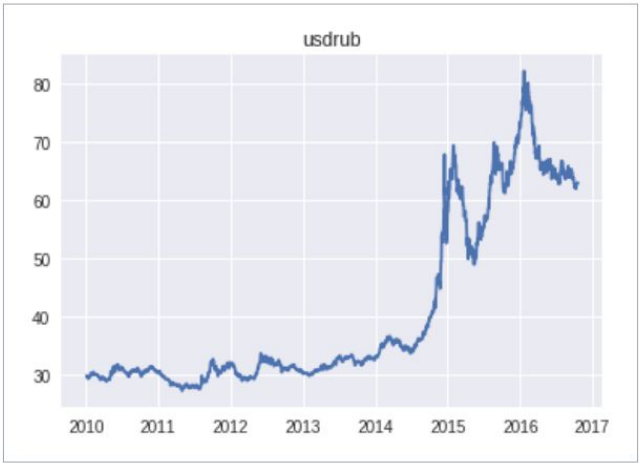
不過除了上面提及的因素以外，我們觀察到似乎還有一些外在的因素導致validation非常不穩定。也就是2015年(當常不完全就是剛好2015年)，只是大概就是似乎有不尋常的經濟情勢使得我們的預測非常不準。這樣推測的原因就是當我們想嘗試尋找russian house price index時查到以下這張圖，也就是說我們預測的2015年，house price並沒有像前幾年一樣正常成長，反而出現負成長





http://www.globalpropertyguide.com/Europe/Russia/Price-History

我們嘗試的一個簡單的實驗，把以前predict結果的submission(還沒有做任何scale down或特別的transform之前的)的前4000到5000筆預測結果直接乘以一個介於0.90到0.99的數字便發現結果都會進步。接著我們就自己把很多macro裡面的data隨時間的變化畫出來，也能發現2015年似乎有發生一些不正常的事，下面附上匯率隨時間的變化圖。




所以相信我們觀察到的這個問題事確實存在的而且是一個很嚴重的因素。但問題就是，照理來說這樣的大環境因素應該要能夠從macro.csv裡面看出來，但嘗試加了

macro.csv裡面的各種feature(的combination)都沒能得到更好的結果。後來查到有人是這麼解釋:

*"When ruble fell down, people tried to save their money, particulary, investing in real estate. At the end of training set they bougth something already and got loans to pay. So at the beginning of test set they have no money an have no interest in buying a flat (already bought)."*

再來就是針對rmlse的觀察:

The RMSLE is calculated as

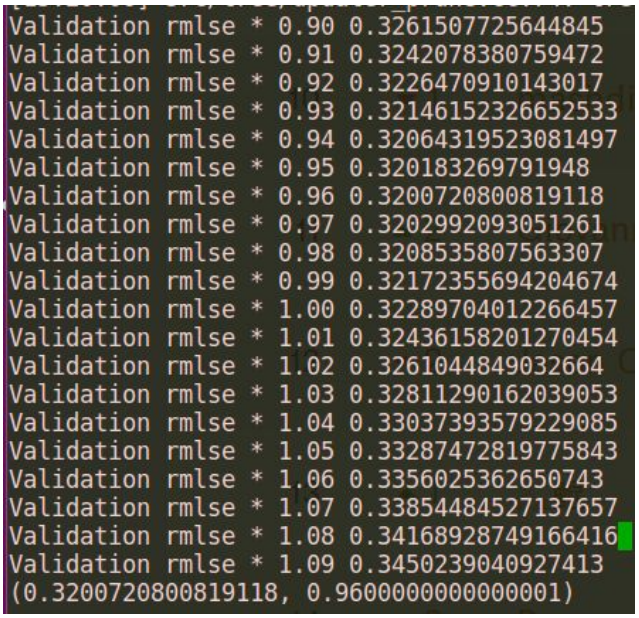
$$\epsilon = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i+1)-\log(a_i+1))^2}$$

Where:

- $\epsilon$ is the RMSLE value (score)
- $n$ is the total sales in the (public/private) data set
- $p_i$ is your prediction
- $a_i$ is the actual sales for $i$
- $\log(x)$ is the natural logarithim of $x$



上傳了一個皆為0的檔案，就可以知道public leaderboard上的答案取log1p後的平方和。再觀察自己predict出來的結果，發現這個值比答案大很多，這有可能是我們的model可能predict出了一些不存在的極值，或者是model本身就偏向預測的比答案還大。所以嘗試做了一個實驗: 看看validation上面是否也會有這種情形。下圖為使用xgboost with early stopping, shuffle後使用sklearn的train_test_split且沒有做額外feature engineering的實驗結果。發現竟然把predict的結果值皆成一個小於1的數字做scale down，validation rmlse竟然就會變小! 做這樣的嘗試之後上傳，分數也有很大的提升。事實上，做這件事之後就過了simple，再tune一下parameters，就可以過strong。




最後我們並沒有使用對結果直接乘一個scale down factor，最後我們採取的方法是把所有值取log1p之後，在對他加一個常數，這樣做的意義其實非常接近於乘一個倍數(ex: 都加上log(0.99) 其實非常接近於對結果乘以0.99)，但這麼做有一個非常大的好處，就是如果我定義我加的常數為C, 那麼rmlse^2其實可以寫成一個C的一元二次方程式，知道一元二次方程式的三個點，我就可以求出取線，對曲線微分即可以求得會使得rmlse最小的C值。

Ensemble of xgboost models

1. Dropping data before 2011, 2012, 2013 or not
2. Train on cleaned data or raw data
3. Add some additional feature engineer
4. Predict log prices or  raw prices
5. Include macro data or not

我們做了許多嘗試，向上述除了三以外的每個選項都有兩種方法。
剛開始，我們使用xgboost，在leaderboard上表現最好的single model結果大約是0.31103。最後我們用四個model，每個model都是上述幾點點不同選擇的組合。做Ensemble後在leaderboard上得到0.31059。不過我們最後最好的結果並不是這個model，因為發現了leaderboard上更高的kernel所以就拿來用(0.31039)，現在在public leaderboard上最好的model是使用那個kernel並調整一下(像是拿掉一些自己覺得沒必要的動作)跑不同random seed做ensemble再用上面提到的方法做transform的結果。

# NN

1. Introduction

問了比較有經驗的老師，得知其實能用deep learning目前在這樣的regression problem上很難擊敗像random forest或gradient boosting這樣的model。畢竟Deep learning approaches have been particularly useful in solving problems in vision, speech and language modeling where feature engineering is tricky and takes a lot of effort. 但當我們發現我們並沒有辦法從macro的data裡面預測出2015年左右那些不尋常的趨勢時(在gradient boosting的討論中所提到的)而必須使用一些magic number做scale down來imporve performance時，就開始想說是不是試一下nn，會不會因為某些feature很複雜我們沒有在feature engineering的時候找出來?

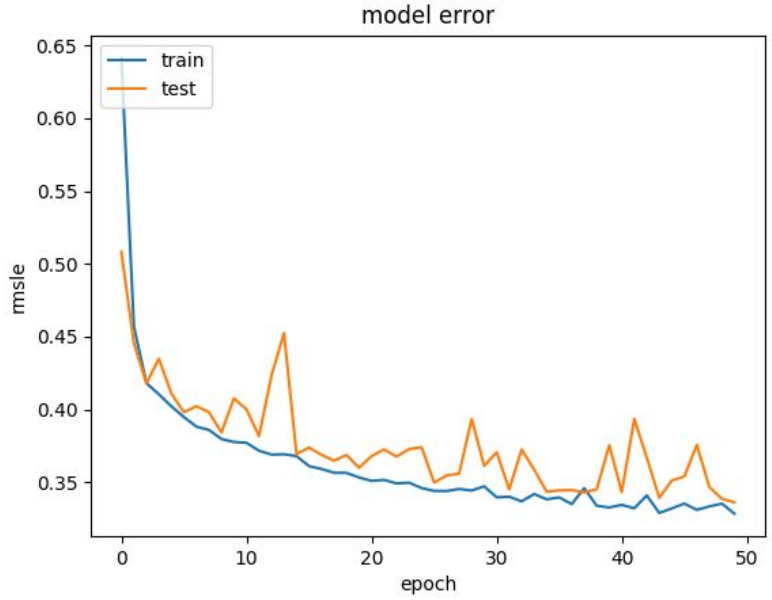
2. Preprocess/ Feature engineering

基本上跟xgb相同，除了對categorical的data使用one hot encoding。

3. Model structure

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 256)               115456
_________________________________________________________________
dense_2 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_3 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 257
=================================================================
Total params: 247,297
Trainable params: 247,297
Non-trainable params: 0
_________________________________________________________________
```

4. Experiments and Discussion




rmsle on training  data: 0.30 ~ 0.33
rmsle on validation data: 0.31 ~ 0.34
Score on kaggle: **0.4x**

訓練過程中，常常跑到一半 loss 就突然到 inf，或是一開始卡在 15 這種很高的數字，我的猜測是 loss function 的圖形太過奇怪了，一開始容易在原地打轉，下降過程中又容易跑到 inf。而在結果方面，看不出有 overfit 或 trainig loss 本身就太高的問題，但在 kaggle 上的分數卻與 validation 相去甚遠。
NN 的另一個問題是訓練的時間。我的模型用 GeForce 840M 跑 100 epochs 大約要兩分鐘，這和 xgboost 用的時間差不多，而在這樣的時間下，xgboost 已經有非常好的結果，雖然可以期望用更複雜許多的 NN 模型可以有比較好的表現，但在效率考量之下，我們便捨棄了這個方法。

## SVR

C = 0.1 validation rmsle = 0.522
C = 1:  validation rmsle = 0.51
C = 50: validation rmsle = 0.518
C = 1000: validation rmsle = 0.59

C是 penalty parameter，越大對誤差容忍度較小，越容易 overfitting。

SVR 光是validation的rmlse都太高了，沒有再做更進一步的嘗試。

## Random Forest Regression

1. Introduction

雖然一般情況下gradient boosting總能outperform random forest，但我們覺得很難確保這個task上對我們沒有幫助，畢竟這兩個model本身就是相對的。Boosting is based on weak learners (high bias, low variance). Boosting reduces error mainly by reducing bias.On the other hand, Random Forest uses as you said fully grown decision trees (low bias, high variance). It tackles the error reduction task in the opposite way: by reducing variance.

2. Preprocessing/Feature engineering
   同xgb

3. Model Structure
   這裡主要使用sklearn的RandomForestRegressor嘗試了
   1.單個RandomForestRegressor
   2.將多個不同參數的RandomForestRegressor做ensemble(直接做平均)
   3.RandomForestRegressor + XGB做ensemble

4. Experient and Discussion

1.單個RandomForestRegressor的參數，除了n_estimators外其他參數調整在validation ser上都沒有明顯的幫助，而n_estimators在超過20後就沒有顯著的提升

| n_estimators | 10(default) | 15 | 20 | 25 |
|---|---|---|---|---|
| Validation rmsle | 0.327 | 0.324 | 0.321 | 0.322 |
| Validation set: random 10% training data | | | | |

2.將n_estimator = 10、15、20的 RandomForestRegressor做ensemble，得到0.3198的validation rmsle，比單個model來的好。

3.RandomForestRegressor + XGB做ensemble的結果介於兩者之間。

Kaggle score:

| Model | RandomForest *1 | RandomForest * 3 | RandomForest + xgb |
|---|---|---|---|
| Kaggle score | 0.33287 | 0.32935 | 0.3142 |

random forest在做ensemble後有效的減少overfitting，在kaggle上的表現有提升，而加上xgb後結果雖然比單個random forest好，但卻比原本的xgb差。random forest所使用的feature跟xgb基本上差不多，也調過參數跟試了ensemble，但做出來結果還是差很多，因此認為在這個data上random forest不是適合的model。

## K-means Clustering + DNN

1. Introduction

除了clustering以外都跟DNN相同，而在clustering時將每筆preprocess之後的training data都視為一多維向量，以此實作K-means分堆，會想這樣做是因為我認為對於此次的資料，分佈極不平均，很難用「一個」model就達到精準預測所有資料的目的。因此，為了解決這個問題，我認為也許可以透過預先分堆的方式，讓彼此環境因素較接近的測資共用一個model，這樣也許就可以獲得比單純的DNN更好的結果。

K-means大致上的作法就是經由不斷的迭代，使得SSE(sum of square error)最小，其公式如下：

$$\sum_{j=1}^{k}\sum_{\mathbf{x}\in C_j} dist(\mathbf{x},\mathbf{m}_j)^2$$

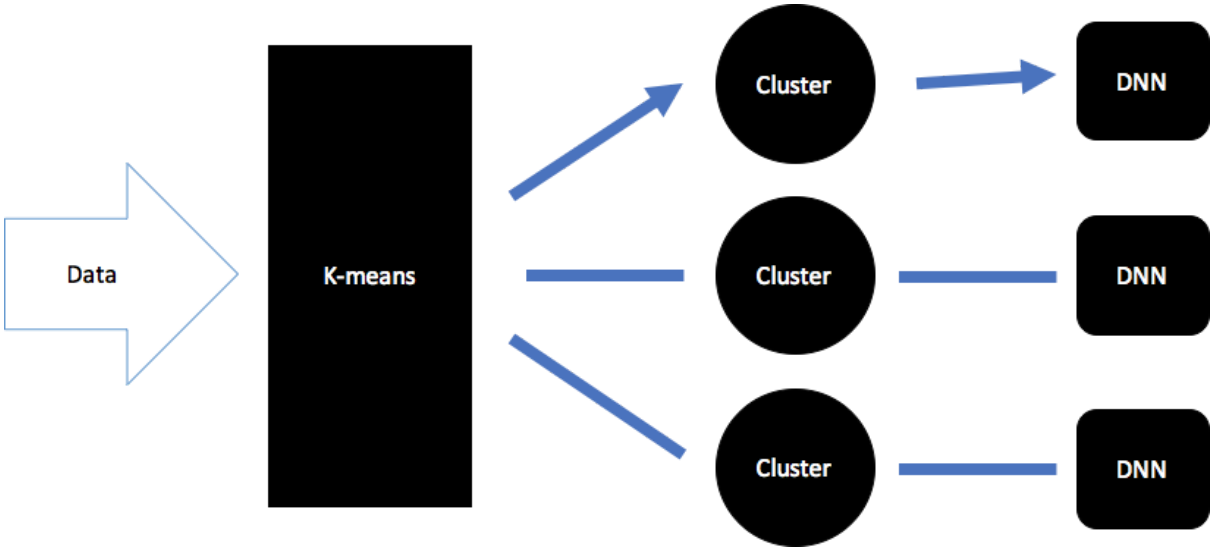
C代表cluster，m代表centroid。
因此，我們可以預期經由k-means clustering能夠得到k堆，且同堆內彼此向量距離相近的資料，而在這裡我假設這些就會是環境因素較相近的資料。

2. Preprocessing/Feature engineering

同nn

3. Model Structure

K-means所fit的corpus是所有的training data，會分成N堆，這N堆會各自訓練自己的DNN model，而這些model的架構就跟DNN一樣，而在預測時就看該筆testing data應該要放在哪一堆裡，就用那個model去預測結果。
其運行架構如下圖




4. Experiments and Discussion

在實驗過程中，其實遇到了一個在上面DNN有提到的問題，訓練過程中常常會跑到inf或是15.多的loss，這邊不再重複解釋可能原因，但這對於K-means Clustering+DNN所造成的影響是因為我會訓練好幾個DNN model，但只要有其中一個卡在那裡，就必須全部重新開始，直到所有model都正常才能得到想要的結果，因此N越大成功的機會就越小了，像我在測試N=12的時候，我嘗試了二三十次才遇到了一次「所有DNN model都正常」的結果。

| N(堆數) | 1 | 3 | 6 | 9 | 12 |
|---|---|---|---|---|---|
| RMSLE | 0.3324 | 0.3249 | 0.3201 | 0.3225 | 0.3194 |

以上的RMSLE都是validation的數值，看起來對於結果有些許的幫助，但傳上kaggle以後的數值卻從原本的0.4x變成0.5x接近0.6，比原來更差了。我想這是因為原本training data跟testing data的環境因素就已經差滿多了，分堆以後反而讓我們的model更符合training data，導致訓練出來的model更不適合用來預測testing data，簡單來說就是我這個方法訓練出了一個overfitting的model。