# Applied Machine Learning HW2 (15%)

Due Wednesday May 16 @ 11:59pm on Canvas

Instructions:

1. In this HW you will compete in an active Kaggle competition, Housing Price Prediction: `https://www.kaggle.com/c/house-prices-advanced-regression-techniques`
2. The objectives of this HW include: understanding and using existing tools for linear and polynomial regression, understanding and using existing tools for regularized linear regression, and (pre-)processing (mixed) categorical and numerical features.
3. This HW should be done in Python, numpy, and sklearn. You can also use other packages such as pandas, though you might find it easier to code in Python yourself.
4. Beside `train.csv` and the semiblind `test.csv` on the website, we also provided the train-dev split (`my_train.csv` and `my_dev.csv`) which is available on the course homepage, so that you can verify your results on dev before submitting test prediction results to kaggle, and so that you can tune hyper-parameters on dev.
5. Part of your grade will be based on your prediction accuracy on test (self report your best prediction error among all your submissions to kaggle).
6. You should submit a single `.zip` file containing `hw2-report.pdf`, `test_submission.csv` (in the format of `sample_submission.csv`), and all your code. Again, LaTeX'ing is recommended but not required.

## 1 Understanding the Evaluation Metric

This kaggle contest uses "Root Mean Squared Log Error" (RMSLE) rather than the standard "Root Mean Squared Error" (RMSE).

1. What exactly is this RMSLE error? (write the mathematical definition).
2. What's the difference between RMSLE and RMSE?
3. Why does this contest adopt RMSLE rather than RMSE?
4. If your RMSLE score is 0.15, what does it mean intuitively, in terms of housing price prediction error?
5. What are your RMSLE error and ranking if you just submit `sample_submission.csv`?
6. What is your "Team name" on kaggle (note this HW should be done individually)?

For the rest of this HW, it is highly recommended that you take the logarithm for the $y$ field (`SalePrice`) before doing any experiment, so that you reduce the problem back to RMSE. However, do not forget to exponentiate it back before submitting to kaggle. In other words, you train your regression systems to predict the logarithm of housing prices, $\log(y)$, but you submit your predictions in the original prices, not the ones after taking log.

## 2 Naive data processing: binarizing all fields

Take a look at the data. Each training example contains 81 fields in total: the unique `Id` field, 79 input fields, and one output field `SalePrice`. Like in HW1 data, there are two types of input fields: categorical, such as `SaleCondition` and `GarageType`, and numerical, such as `LotArea` and `YrSold`. Note that some fields might be mixed categorical/numerical, such as `LotFrontage` and `GarageYrBlt`: as you know, not all homes have lot frontages (the length of the front side of the lot facing a street – some lots are not facing any street), and not all homes have garages, thus both fields could have occasional `NA` values (for "N/A" or "not applicable"). Following HW1, the simplest thing to do is to binarize all fields. You can use your own Python implementation, or `pandas.get_dummies()`, ~~or `sklearn.preprocessing.MultiLabelBinarizer()`~~, but I think it is best to use your own implementation for its flexibility and for the rest of this HW.

1. How many features do you get? (Hint: ~~around 4380~~ **around 7230**).

   **You can use this command to see the total number of binary features:**

   ```
   for i in `seq 2 80`; do cat my_train.csv | cut -f $i -d ',' | sort | uniq | wc -l; done | \
   awk '{s+=$1-1} END {print s}'
   ```

   Let me explain a little bit: the first line loops over each column (except the first which is Id and the last which is the output), and print the values for that column (`cut -f $i`), sort and make unique, and count how many unique values there are for that column (`wc -l`). The second line uses `awk` to sum it up; here `$1` denotes the first column, and `-1` for excluding the header row, and finally print the sum.

   **The reason why the TAs incorrectly suggested "4380" features initially was due to the use of `sklearn.preprocessing.MultiLabelBinarizer`, and we now recommend you NOT to use it. As I said, implementing binarization in Python yourself is always the safest and most flexible way. Thanks a lot to Samantha for pointing this out!**

2. How many features are there for each field?

3. Do you need to augment the space (add bias dimension) like in HW1, or does your regression tool automatically does it for you?

4. Train linear regression using `sklearn.linear_model.LinearRegression` or `np.polyfit` on `my_train.csv` and test on `my_dev.csv`. What's your root mean squared log error (RMSLE)? (Hint: should be around ~~0.22~~ **improved to 0.15 or 0.16 after the bug fix**)

5. What are your top 10 most positive and top 10 most negative features? Do they make sense?

6. What's your feature weight for the bias dimension? Does it make sense?

7. Now predict on `test.csv`, and submit your predictions to the kaggle server. What's your score (RMSLE) and ranking?

## 3 Smarter binarization: Only binarizing categorical features

You might have observed that most numerical features shouldn't have been binarized: for example, features like `LotArea` are always positively correlated with the sale price.

1. What are the drawbacks of naive binarization? (Hint: data sparseness)

2. Now binarize only the categorical features, and keep the numerical features as is. What about the mixed features such as `LotFrontage` and `GarageYrBlt`?

3. Redo all the questions asked in the naive binarization section. (Hint: the new dev error should be around 0.14, which is much better than naive binarization).

You will see that even if you just do everything up to this point, you should be ranked reasonably in this contest.

## 4 Experimentation

Try the following to improve your score.

1. Try regularized linear regression (`sklearn.linear_model.Ridge`). Tune $\alpha$ on dev. Should improve both naive and smart binarization by a little bit.

2. Try non-linear features: what if the sale price is quadratically correlated with some of the most important numerical features such as `OverallArea` (also known as square footage) and `LotArea`?

3. Try feature combinations. An obvious candidate is adding a new feature `Years_since_remodeled` = `YearRemodeled - YearBuilt`. But is a feature like this really useful?

4. BTW, how are these non-linear features (including feature combinations) relate to non-linear features in the perceptron? (think of XOR)

5. Try anything else that you can think of. You can also find inspirations online, but you have to implement everything yourself (you are not allowed to copy other people's code).

What's your best dev error, and what's your best test error and ranking? Take a screen shot of your best test error and ranking, and include your best submission file.

**Debriefing (required):**

1. Approximately how many hours did you spend on this assignment?
2. Would you rate it as easy, moderate, or difficult?
3. Did you work on it mostly alone, or mostly with other people?
4. How deeply do you feel you understand the material it covers (0%–100%)?
5. Any other comments?