AI 534 Machine Learning HW3 (15 pts; **extra +2 pts**)

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`.
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`) in `hw3-data.tgz` which is available on Canvas and at `https://classes.engr.oregonstate.edu/eecs/fall2023/ai534-400/unit3/hw3/hw3-data.tgz`, 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 `hw3-report.pdf`, `test_submission.csv` (in the format of `sample_submission.csv`), and all your code. Again, LaTeX'ing is recommended ~~(+1)~~ but not required.
7. Please read: `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html`. See also `https://inria.github.io/scikit-learn-mooc/python_scripts/linear_regression_non_linear_link.html`.

1 Understanding the Evaluation Metric (1.5 pts)

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). (0.5 pts)
2. What's the difference between RMSLE and RMSE? (0.25 pts)
3. Why does this contest adopt RMSLE rather than RMSE? (0.25 pts)
4. One of our TAs got an RMSLE score of 0.11 and was ranked 28 in Spring 2018. What does this 0.11 mean intuitively, in terms of housing price prediction error? (0.25 pts)
5. What are your RMSLE error and ranking if you just submit `sample_submission.csv`? (0.25 pts)
6. What is your "Team name" on kaggle (note this HW should be done individually)? (0 pts)

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 convert the objective back to RMSE (i.e., optimizing RMSE of $\log y$ is equivalent to optimizing RMSLE of y). 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 logarithm. **Hint: in machine learning, you should always use `np.log()` instead of `math.log()` because the former works with vectors and matrices (same for `exp()`, `sum()`, etc.)**

Extra credit question (+1 pt): Why do you need to do this?

2 Naive data processing: binarizing all fields (4.75 pts)

Take a look at the data. Each training example contains 81 fields in total: the unique `Id` field, 79 input fields **x**, and one output field $y =$ `SalePrice`. Like in HW1 data, there are two types of input fields:

1. categorical, such as `SaleCondition` and `GarageType`, and
2. numerical, such as `LotArea` and `YrSold`.

Note that some fields might be mixed categorical/numerical, such as `LotFrontage` and `GarageYrBlt` because 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"). **Hint: you can convert all fields to strings first, e.g.,** `data = data.astype(str)` **.** Following HW1, the simplest thing to do is to binarize all fields (using sklearn).

1. How many features do you get? (Hint: around 7,230). (0.75 pts)

   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.

2. How many features are there for each field? (0.5 pts)

3. 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) on dev? (Hint: should be **∼0.152**). (1 pt)

4. What are your top 10 most positive and top 10 most negative features? Do they make sense? (1 pt)

5. Do you need to add the bias dimension (i.e., augmented space) explicitly like in HW2, or does your regression tool automatically handle it for you? Hint: `coef_` and `intercept_`. What's your feature weight for the bias dimension? Does it make sense? (0.5 pts)

   Extra credit question (+1 pt): What's the intuitive meaning (in terms of housing price) of this bias feature weight?

6. Now predict on `test.csv`, and submit your predictions to the kaggle server. What's your score (RMSLE, **should be around 0.16**) and ranking? (1 pt)

3 Smarter binarization: Only binarizing categorical features (3.5 pts)

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, etc.) (0.5 pts)

2. Now binarize only the categorical features, and keep the numerical features as is. What about the mixed features such as `LotFrontage` and `GarageYrBlt`? (1 pt)

3. Redo the following questions from the naive binarization section. (Hint: the new dev error should be around 0.14, which is much better than naive binarization). (2 pts)

(a) How many features are there in total? (0.25 pts)

(b) What's the new dev error rate (RMSLE)? (0.5 pts) **(should be ∼0.12)**

(c) What are the top 10 most positive and top 10 most negative features? Are they different from the previous section? (0.5 pts)

(d) Now predict on `test.csv`, and submit your predictions to the kaggle server. What's your score (RMSLE, **should be ∼0.13**) and ranking? (0.75 pts)

You will see that even if you just stop at this point, you should be ranked reasonably in this contest. **Hint: you might need to normalize numerical features like you did in HW1.**

4 Experimentation (5.25 pts)

Try the following to improve your score and ranking.

1. Try regularized linear regression (`sklearn.linear_model.Ridge`). Tune α on dev. Should improve both naive and smart binarization by a little bit. (0.5 pts)

2. Try non-linear regression (`sklearn.preprocessing.PolynomialFeatures`) (1 pt).

   The sale price might be non-linearly correlated with some most important numerical features such as `OverallArea` (also known as square footage) and `LotArea`, or some feature combination. Instead of creating such non-linear features (including combinations), you can use any of the following three methods:

```
>>> import numpy as np
>>> X = np.arange(6).reshape(2, 3); X
array([[0, 1, 2],
       [3, 4, 5]])
>>> XX = np.concatenate([X, X**2], axis=1); XX # method 1: manual non-linear features
array([[ 0,  1,  2,  0,  1,  4],
       [ 3,  4,  5,  9, 16, 25]])
>>> from sklearn.preprocessing import PolynomialFeatures # method 2: PolynomialFeatures
>>> PolynomialFeatures(degree=2).fit_transform(X) # same result as method 1
array([[ 1.,  0.,  1.,  2.,  0.,  0.,  0.,  1.,  2.,  4.],
       [ 1.,  3.,  4.,  5.,  9., 12., 15., 16., 20., 25.]])
>>> from sklearn.pipeline import make_pipeline # method 3: pipeline + PolynomialFeatures
>>> polynomial_regression = make_pipeline(
      PolynomialFeatures(degree=2, include_bias=False),
      LinearRegression())) # "include_bias=False": no explicit bias term "1" (cf. intercept_)
>>> polynomial_regression.fit(data, target) # data is input labels, target is output label
>>> target_predicted = polynomial_regression.predict(data)
>>> mse = mean_squared_error(target, target_predicted)
```

   See also `https://inria.github.io/scikit-learn-mooc/python_scripts/linear_regression_non_linear_link.html`.

3. How are these non-linear features (including feature combinations) relate to non-linear features in the perceptron? (think of XOR) (0.25 pts)

4. 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). (0.5 pts)

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. (3 pts)

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?