CS 181 Spring 2017
Practical 1 Writeup
Team: DuckDuckGoose
Alexander Lin, Melissa Yu, Matthew Huang

1 Technical Approach

Our approach throughout this practical was to start simple, using convenient data and simple, fast-training models. By seeing which simple models worked better, we added layers of complexity via alternative data sources and larger, more involved models. As our goal is to beat linear regression, our starting point was to run linear regression on the given 256-size bit feature vector. After implementing this baseline, we immediately noticed an issue with the data: most features of the 256-size bit vector were either 0 or 1 (the variance of the features were 0) — that is, most features gave no information in the train set, and therefore would not be helpful for us. We extracted the 31 features that actually had variation among the different molecules, putting the others aside.

After verifying that a simple linear regression model on these 31 features gave the same r-squared value as the LinReg model on 256 features, we ran ridge regression, a natural followup. Since the value of λ is crucial in ridge regression, we ran cross-validation on the train set to determine an appropriate estimate for λ, and more importantly, a reliable estimation of how well our model performs relative to the baseline.

After ridge regression, we ran a simple neural network with one hidden layer, as a preliminary indicator of how well a neural net might perform on our data set. Given that we are only working with 31 features, it was very plausible that a neural network would be able to capture more nuanced interactions between our features (due to the activation function in the hidden layer). However, a simple neural-network did not improve performance enough, as we only saw an increase of ~ 0.07 in the r-squared value, as opposed to naive linear regression. At this point, our model was still performing worse than the random forest classifier!

It was clear that either 31 features were simply not enough, or we needed more power and complexity to capture the interactions between different interactions. The most straightforward and feasible plan of action was to consider the interaction between pairs of bit features. Given two bit features b_1, b_2, one way to capture their interaction would be to take a bitwise and, $b_1 b_2$. However, as shown in the left-hand logical truth table below, on average, 3/4 of the feature values would be 0, while only 1/4 would be 1 (assuming $p(b_i = 1) = 0.5$). With XOR, 1/2 would be 0, and the other 1/2 would be 1. As we can see, XOR features have higher entropy, and would convey more information with their bit.

AND	$b_1 = 0$	$b_1 = 1$
$b_2 = 0$	0	0
$b_2 = 1$	0	1

XOR	$b_1 = 0$	$b_1 = 1$
$b_2 = 0$	0	1
$b_2 = 1$	1	0

Therefore, we proceeded with the XOR approach, as conveying more information per bit is crucial, especially when we are analyzing the interactions between pre-existing. We thus

generated $\binom{31}{2} = 465$ features, which are simply pairwise XORs of the 31 useful bit features. We used these $465 + 31$ features to run some preliminary simple models: ridge regression and a small neural network. Though small improvements were evident, the r-squared value of the best model was still comparable to that of the random forest baseline classifier, so we explored other paths of improvement.

We realized that we needed more data: our 31 initial features are bits and encode the minimal amount of information possible. Further, even after including feature XOR interactions, our model was not as accurate as we liked. We believe that there is simply not enough data encoded in the 31 features, and that we would be better off finding data from alternative sources.

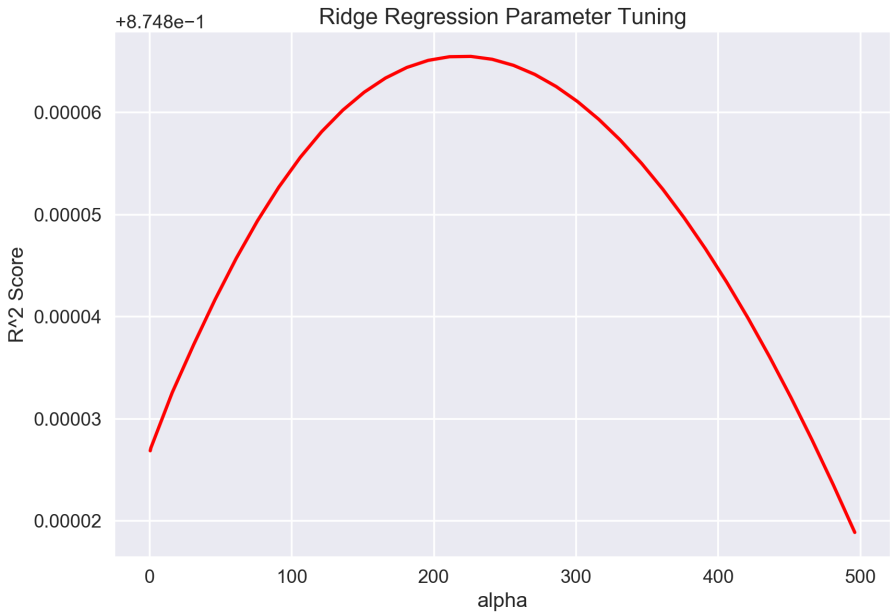
Per suggestion of the practical guidelines, we explored RDKit to see other molecular features that we could extract. We decided to use a function that extracts "fingerprints" of molecules based on their structure, in the form of a 2048-bit vector. For instance, one fingerprint bit might correspond to one fragment of the molecule, so molecules that were structurally similar would have similar fingerprint vectors.

This is a very promising feature—each bit represents the presence of a particular structural element, and therefore elicits properties of the molecule. Combining these into a 2048-bit vector can provide a large amount of data for our disposal. Further, it is even more imperative now to encode the interaction between bits: the presence of structural elements s_1 and s_2 by themselves may not be significant, but a molecule that possesses both elements may take on a different property. This suggests that armed with our new set of 2048 features, a neural network holds high potential.

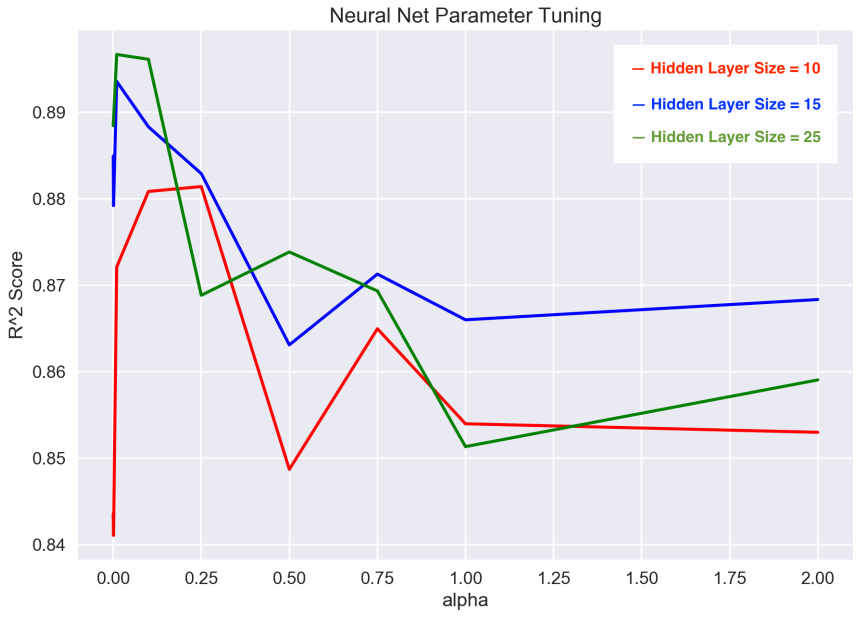
After a simple preliminary train of a neural network on our features with no parameter tuning, our r-squared value increases to over 0.90, versus 0.88 for linear regression and 0.88 for un-tuned random forest. This shows that we are on the right track—the next step is to tune our hyperparameters. Using a simple k-fold cross-validation with $k = 5$, we tune various hyperparameters, including the number of layers, the number of nodes in each layer, the activation function, and the underlying class of regression used. We considered a ridge regression to use in the neural net, and the simple rectified linear activation function.

2 Results

We present data on the tuning of hyperparameters training on the 2048 bit features. After running a simple k-fold cross validation with $k = 5$ on the ridge regression with various values of λ, we can see that the average validation set error is not very high, peaking at $R^2 = 0.88$. Moreover, although there is a clear choice of $\lambda \approx 200$ which maximizes the accuracy of our model, the range of the R^2 values covered in the set of regularization strengths tested is less than 0.01%.



For the Neural Net, computing power restrictions forced us to hold the number of hidden layers constant at 1 (see *Discussion* for more details). A few cross-validation experiments showed us that the simple 'relu' activation function was fast and performed well compared to other functions. Thus, we tuned 2 parameters using k-fold cross-validation with $k = 5$: α, the L2 penalty (regularization term) parameter, and s, the size of the single hidden layer. As shown in the graph below, the optimal result of $R^2 = 0.92$ was achieved with a hidden layer of size 25 and $\alpha \approx 0.1$.



The table below shows the k-fold R^2 scores for several runs of each model with the optimal parameters found earlier. Comparing performance of the main methods used in this practical on both the original feature set and the molecular fingerprint features, we see that the combination of Neural Network and molecular fingerprints perform best. On the unlabeled testing set, our algorithm achieves a reasonably good score of 0.12 (RMSE).

Model\Features	Original	Original + XOR	Fingerprint
Linear Regression	0.46	0.48	0.88
Ridge Regression	0.47	0.52	0.88
Neural Network	0.55	0.56	0.92

3 Discussion

During the course of this project, the limited computing power of our laptops was the largest barrier. Given the large size of the data, simply loading and converting the original training data to "fingerprint" representations took an hour alone! More importantly, after performing feature extraction on the data, training any moderately complex model was extremely time consuming. While training the Neural Net, we expected larger numbers of hidden layers to give us better generalization abilities, but using because the runtime grows exponentially with the number of layers, using more than 1 hidden layer in our code caused our computers to kill the process due to excess memory usage. A few techniques our team identified to improve our memory and time performance were (1) Processing data in smaller chunks during feature extraction and model training (if warm-start for the model was possible), (2) Explicitly declaring the smallest possible data type to use (instead of letting python infer the data type) when reading in objects from memory to avoid wasted space, and (3) Training on only a portion of the available training data.

It's worth noting that Gaussian process Regression (GPR) has also been shown in research to yield good regression results for the HOMO-LUMO gap in conjunction with the molecular fingerprint representation of the SMILES string (Sun, Hong Yang. 2014. *Learning over Molecules: Representations and Kernels.*). The Gaussian process assumes that the distribution of the function $y = f(x)$ for a set of points X is jointly Gaussian, with a covariance matrix modeled by a kernel function K which measures the similarity between vectors x_i and x_j. Intuitively, this approach should work well with data given a reasonable choice of kernel function. Using our own homemade implementation of the simple Tanimoto similarity metric kernel, we tested this approach and found that it yielded promising results ($R^2 = 0.87$) without any hyperparameter tuning. However, because fitting a Gaussian Process requires inverting the kernel matrix, the algorithm run time scales as $O(N^3)$, where N is the training set size. Coupled with our homegrown implementation of the kernel, just one run of the algorithm on 1% of the training data took over 2 hours to complete, making it infeasible for us to further explore this promising avenue.

Note that Random Forest also presents promising results, since an un-optimized run of the algorithm yielded a R^2 of 0.88. This is another direction for future work.