Practical 1: Predicting the Efficiency of Organic Photovoltaics

Team Name: ☆☆☆ DRAGON ARMY ☆☆☆
Team Members: Brandon Sim, Jeremy Nixon, Lydia Chen

Introduction

This paper describes our approach to predicting HOMO-LUMO gaps for photovoltaic molecules using a machine-learning based approach, rather than traditional quantum chemical calculations, which may be prohibitively expensive to run on a large set of molecules. We consider a variety of regression techniques, and dive especially deeply into the random forest regressor. We also generate a variety of additional features to improve the quality of our predictions, and report our results for various feature sets and classifiers. We obtain a final RMSE of around 6.6%.

Linear Regressors

We began our approach with a survey of various linear regressors, including various regularization techniques (L1, L2, L1 and L2). We trained the default feature set using a ridge regression, a LASSO regression, and a combination of these regularization techniques, the elastic net. We performed 10-fold cross validation on a variety of α values for the ridge regression as well as the LASSO regression. The results are summarized in the results section. However, because of limited computational time, we decided to focus on one type of regressor, the random forest regressor, and did not test linear regressors with any regularization type in depth. In the future, we would like to consider elastic nets in particular due to their ability to generate sparse solutions (without the drawbacks of pure LASSO) on the final feature set that we generated.

Random Forest Regressor

Given the nature of the input data, a regression tree is an attractive method for this particular machine learning problem. Regression trees are invariant to transformations of the input data, which in this case is largely binary with some exceptions in the features we selected later. At each node, the tree considers a random sample K of the features. The node chooses, out of the K features under consideration, the feature that minimizes the sum of squared errors in splitting the data into two more nodes. This process is repeated at each node until each node contains one data point. The decision tree is resilient to features with poor predictive power because even if all K features considered at a particular node are useless in predicting the target value, the tree will simply make a split as described above and the resulting subtrees will (likely) be split on critical predictors as the tree grows deeper [5]. This allows us to focus on generating and testing groups of different types of features, rather than selecting for or excluding individual features. For most purposes $K = \sqrt{M}$ where M is the number of features is a computationally efficient and sufficiently large number for N [5]. We performed a cross-validation test and actually found that considering a random sample of $\sqrt{M}$ features slightly outperformed the default of considering all M features at each node, both in runtime and accuracy.

However, a single tree is rarely accurate, since there is the chance that a tree just happens to consider poor predictors at every node, or that certain predictors overfit particular subsets of the data. To address the problem of overfitting we used a random forest, which trains N trees on samples of the training data and then averages the predictions from each of the N trees, which decreases the bias that often occurs in one tree. Part of the improvement that comes from using random forests instead of single regression trees is that each tree is trained on random bootstrapped samples of the training data, allowing for randomness that will be averaged out over the complete forest. Our final prediction $\hat{y}$ can be expressed as the average of the predictions of N regression trees $\hat{y}_n$.

$$\hat{y} = \frac{1}{N}\sum_{n=0}^{N} \hat{y}_n$$

As expected, the number of trees N is a particularly important parameter to tune for random forests. Two few trees can result in high variance and less accuracy on predictions. Too many trees can significantly slow down computing. On average, the runtime of a random forest is quasilinear in the number of trees, $O(NM \log N)$ [3]. In practice, however, running the forests on larger features sets showed a more than linear increase in runtime. We tuned this parameter by cross-validating random forests of varying sizes on the original feature set. The results can be seen in Figure 1.

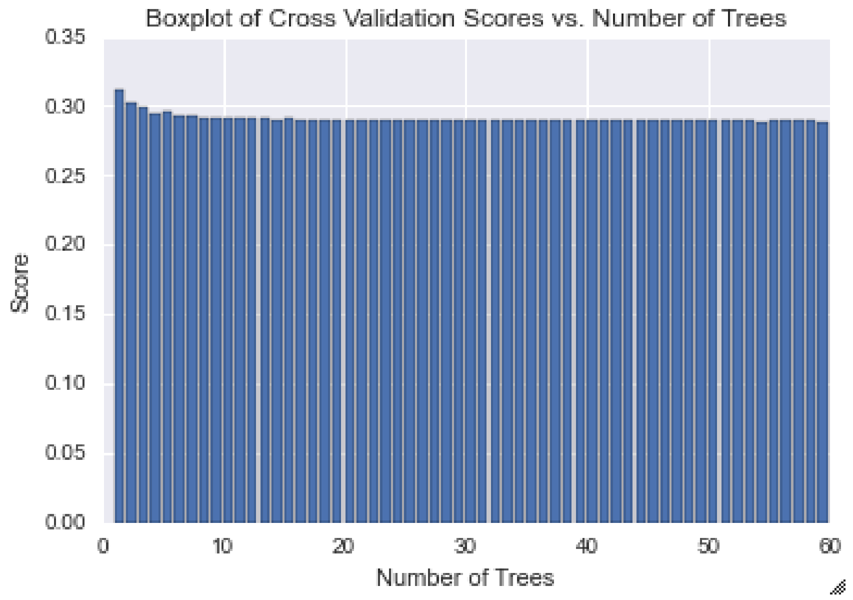

Figure 1: Mean absolute error for 256 features with different sizes of random forest

Ignoring the slight noise due to randomness, the marginal improvement in predictive power from adding trees to the random forest flattens out around 30 trees, although mean absolute error continues to decrease as the number of trees increases.

Feature Selection

Using rdkit, we generated several new sets of features, which are summarized in Table 1.

Feature Type	**Description**	**Parameters**
Morgan/Circular Fingerprint	Uses the Morgan algorithm to generate a bit vector for each atom and its surrounding neighborhood (defined by parameter radius)	Radius, Number of Bits, Features Invariant
Topological Fingerprint	Identifies and hashes bond paths to generate a bit vector	Number of Bits
MACCS Keys	166 pre-identified substructures used to generate a fingerprint	None
Chemical Properties	Various binary and nonbinary chemical descriptors for molecules	Features

Table 1: Summary of various fingerprints and features generated

General Strategy

Our general strategy with feature selection was to generate a large variety of features, then decide quantitatively through cross-validation which features would be most useful for inclusion in our final model. Below we discuss several classes of features that we generated and considered; our final submission, as discussed more in depth in our results section, uses a combination of the Morgan (with and without features) fingerprints and the chemical descriptor features.

Morgan/Circular Fingerprint

The Morgan algorithm generates a bit vector in a given radius around each atom of a molecule. The generated vectors are then hashed to a vector of a user-specified length.

Radius: The radius parameter defines the size of the atom environment that the algorithm considers when considering each individual atom. A radius of 1, for example, considers the atom and atoms one chemical bond away. Literature suggests that a radius of 2-3 is standard, and through our own cross-validation tests we found that radii above 5 did not significantly improve predictive power.

Number of bits: Since the Morgan algorithm hashes the results of the analysis of each atom to a bit vector of a specified length, the probability of collision will decrease with a larger number of bits, meaning that we will get a more descriptive fingerprint. We tried vectors of length 256 (as given), 512, and 1024, with cross-validation tests confirming that more bits resulted in better predictions (albeit with decreasing marginal returns). The limiting factor on tuning this parameter was computing power.

Feature-invariant versus atom-invariant: A parameter that ended up being key to our predictive power was toggling the feature-invariant parameter in rdkit. The first runs we did had the feature-invariant turned on, meaning that the Morgan algorithm looked for pre-defined features based on chemical standards that have proven, in general, to be useful at differentiating molecules. On the other hand, the atom-invariant approach looks for different connectivity information for each atom being considered–number of rings, number of attached hydrogens, etc.

As noted in the rdkit documentation [2], these two approaches (feature-invariant, atom-invariant) can sometimes lead to very different similarity scores between the same molecules, and we found that in this case the features that were generated were at least partially orthogonal; in cross-validation tests we saw a marked improvement by concatenating both sets of features and using them as our predictors as opposed to either set alone. The results suggest that the HOMO-LUMO gap is best predicted by some combination of feature and atom invariant features.

Topological Fingerprint

Topological fingerprints identify and hash topological paths to bit vectors. Therefore, as with Morgan Fingerprints, longer bit vectors decrease the probability of collision and allow for more useful fingerprints. The topological features that were identified did not end up outperforming the Morgan features and also did not seem to add any additional predictive power when concatenated with the Morgan feature sets.

MACCS Keys

Much like the feature-invariant approach in the Morgan Fingerprint, MACCS Keys uses a set of 166 pre-identified substructures as features [1]. As it turns out, the rdkit documentation notes that the MACCS Keys and the Morgan feature invariants are based on similar sets of substructures, and so it is not surprising that MACCS failed to outperform even when concatenated with other feature sets.

Chemical Properties

We considered a variety of chemical descriptors, which includes properties calculated about a molecule based on its molecular structure. Some of these descriptors may have been partially or completely encoded in hashed form via some of the other fingerprint-based methods that we tried, and this correlation may have led to some bias; however, overall, we found that the addition of these chemical descriptor features yielded improvements to our model. The full list of chemical descriptors we considered is listed in this RDKit documentation link. From this list, we extracted

183 features, some of which include molecular weight, number of rings, and various other surface area-related features, such as approximate van der Waals surface area and topological polar surface area. A full list of the features we used can be found in the appendix. The types of data in this feature set, as seen in the appendix, took on a range of values, ranging from binary to integer (such as number of rings, etc.) to continuous variables (such as molecular mass or VSA).

Because the rest of our data was binary, we performed LASSO regressions in an attempt to perform feature selection, although we were somewhat confident that our random forest regressor would not overfit or run into huge problems with a mix of continuous and binary data. We also performed this analysis to attempt to see if the most important features as chosen by regularization were reasonable, as these chemical descriptors were more easily mapped to quantities that could be intuitively understood to have some effect on the HOMO-LUMO gap (unlike the fingerprinting results, where the hash made it difficult to tell which feature was impacting which bit). We performed regularization at various α levels, using LASSO due to its shrinkage properties, and found various subsets of the features which seemed to be most important. Specifically, we found a certain subset of features to be useful at the $\alpha = 0.01$ level, which is found in the appendix here. We see that some of the most indicative features are the various van der Waals surface area calculations, which make sense intuitively as the size of the surface area in a van der Waals sense help govern the bonding properties and thus the HOMO-LUMO gap. However, as we are not chemists, we simply accept this preliminary eyeball test; in our final run, however, we ended up using all 183 features, as the random forest regressor seemed to be adequately not overfitting the data, even with extraneous features (seen in the table below). We include this discussion of features anyway because the selection of these features as important may lead to some chemical understanding of important features to look at, or may inspire further generation of other similar features which may prove useful even in machine learning approaches.

RMSE (100k training) (Mean over 5 runs each)	*Number of Trees*			
Chem Features	30	40	50	60
None	n/a	0.120441	0.119135	n/a
$\alpha = 0$ (all)	0.116613	0.115653	0.114607	0.114128
$\alpha = 0.001$	0.116864	n/a	n/a	n/a
$\alpha = 0.01$	0.117339	n/a	n/a	n/a
$\alpha = 0.1$	0.117965	n/a	n/an	n/a

Table 2: RMSE results for various regularization parameters on chemical descriptor feature set

Results

Below we present a comprehensive table which document some (but not all) of the training and validation that we considered when deciding which paths to pursue during this practical. Of key importance was the lack of computing power of our team; processing took place on two laptop machines throughout the practical (only two of the laptops owned had sufficient RAM to load the data necessary). Given this constraint, many entries in the testing matrix below are blank due to time constraints. Because of theoretical reasons discussed above, we decided to hone in on one regression model, the random forest regressor, and focus on feature engineering and selection.

Here, we present RMSE results in tabular form: across we have various feature sets, and vertically we used various types of regressors. The feature set letters correspond to the following list (separated for readability):

A: Morgan fingerprints, radius 1, 256-bit, with features

B: Topological fingerprinting, 1024 bits

C: Morgan fingerprints, radius 2, 512-bit, with features

D: Morgan fingerprints, radius 3, 512-bit, with features

E: Morgan fingerprints, radius 3, 1024-bit, with features

F: Morgan fingerprints, radius 3, 512-bit, no features

G: Combination: Morgan fingerprints, radius 3, 512-bit, features *and* no features

H: Combination: Morgan fingerprints, radius 3, 512-bit, features *and* no features; *and* chemical features

Results	*Feature Set (across)*							
Method (down)	A	B	C	D	E	F	G	H
Mean	0.40682							
OLS Regression	0.29845							
SGD Regression (out of the box)	0.29857							
SGD Elastic Net (L1+L2)	0.29863							
Support Vector Regression	0.29728							
Random Forest Regression ($K = n_{\text{features}}$), 30 trees	0.27213							
Random Forest Regression ($K = \sqrt{n_{\text{features}}}$), 30 trees	0.27205	0.13009	0.11343	0.10152	0.09896	0.08201	0.07056	0.06765
Random Forest Regression ($K = \sqrt{n_{\text{features}}}$), 60 trees								0.06566*

Table 3: Matrix of RMSE results for various feature sets and regressors; (*) is the final submission

As can be seen in the above table, we tried various regressors, but decided to focus on the random forest regressor and tuning parameters for that regressor due to limited computational power and due to the attractive features of the random forest as discussed in the body of the paper. Through feature selection improvements, we make gains and eventually come to an RMSE of around 6.6%. This result held up in the test set, lending credence to our belief about random forests being resilient to overfitting.

References

[1] Dalke, Andrew. MACCS key 44. Scientific AB, 2013.

[2] Landrum, Greg. The RDKit Documentation. 2014.

[3] Loupe, Gilles. Understanding Random Forest: From Theory to Practice. University of Liege, 2014.

[4] Pedregosa, Fabian, et al. Scikit-learn: Machine Learning in Python. JMLR 12 pp. 2825-2830, 2011.

[5] Steinberg, Dan, Mikhail Golovnya, and N. Scott Cardell. A Brief Overview to Random Forests. Salford Systems, 2004.

Appendix

BalabanJ	NumAromaticHeterocycles	SlogP_VSA5	fr_alkyl_halide	fr_phenol_noOrthoHbond
BertzCT	NumAromaticRings	SlogP_VSA6	fr_allylic_oxid	fr_phos_acid
Chi0	NumHAcceptors	SlogP_VSA7	fr_amide	fr_phos_ester
Chi0n	NumHDonors	SlogP_VSA8	fr_amidine	fr_piperdine
Chi0v	NumHeteroatoms	SlogP_VSA9	fr_aniline	fr_piperzine
Chi1	NumRotatableBonds	TPSA	fr_aryl_methyl	fr_priamide
Chi1n	NumSaturatedCarbocycles	VSA_EState1	fr_azide	fr_prisulfonamd
Chi1v	NumSaturatedHeterocycles	VSA_EState10	fr_azo	fr_pyridine
Chi2n	NumSaturatedRings	VSA_EState2	fr_barbitur	fr_quatN
Chi2v	PEOE_VSA1	VSA_EState3	fr_benzene	fr_sulfide
Chi3n	PEOE_VSA10	VSA_EState4	fr_benzodiazepine	fr_sulfonamd
Chi3v	PEOE_VSA11	VSA_EState5	fr_bicyclic	fr_sulfone
Chi4n	PEOE_VSA12	VSA_EState6	fr_diazo	fr_term_acetylene
Chi4v	PEOE_VSA13	VSA_EState7	fr_dihydropyridine	fr_tetrazole
EState_VSA1	PEOE_VSA14	VSA_EState8	fr_epoxide	fr_thiazole
EState_VSA10	PEOE_VSA2	VSA_EState9	fr_ester	fr_thiocyan
EState_VSA11	PEOE_VSA3	fr_Al_COO	fr_ether	fr_thiophene
EState_VSA2	PEOE_VSA4	fr_Al_OH	fr_furan	fr_unbrch_alkane
EState_VSA3	PEOE_VSA5	fr_Al_OH_noTert	fr_guanido	fr_urea
EState_VSA4	PEOE_VSA6	fr_ArN	fr_halogen	
EState_VSA5	PEOE_VSA7	fr_Ar_COO	fr_hdrzine	
EState_VSA6	PEOE_VSA8	fr_Ar_N	fr_hdrzone	
EState_VSA7	PEOE_VSA9	fr_Ar_NH	fr_imidazole	
EState_VSA8	RingCount	fr_Ar_OH	fr_imide	
EState_VSA9	SMR_VSA1	fr_COO	fr_isocyan	
FractionCSP3	SMR_VSA10	fr_COO2	fr_isothiocyan	
HallKierAlpha	SMR_VSA2	fr_C_O	fr_ketone	
HeavyAtomCount	SMR_VSA3	fr_C_O_noCOO	fr_ketone_Topliss	
Ipc	SMR_VSA4	fr_C_S	fr_lactam	
Kappa1	SMR_VSA5	fr_HOCCN	fr_lactone	
Kappa2	SMR_VSA6	fr_Imine	fr_methoxy	
Kappa3	SMR_VSA7	fr_NH0	fr_morpholine	
LabuteASA	SMR_VSA8	fr_NH1	fr_nitrile	
MolLogP	SMR_VSA9	fr_NH2	fr_nitro	
MolMR	SlogP_VSA1	fr_N_O	fr_nitro_arom	
NHOHCount	SlogP_VSA10	fr_Ndealkylation1	fr_nitro_arom_nonortho	
NOCount	SlogP_VSA11	fr_Ndealkylation2	fr_nitroso	
NumAliphaticCarbocycles	SlogP_VSA12	fr_Nhpyrrole	fr_oxazole	
NumAliphaticHeterocycles	SlogP_VSA2	fr_SH	fr_oxime	
NumAliphaticRings	SlogP_VSA3	fr_aldehyde	fr_para_hydroxylation	
NumAromaticCarbocycles	SlogP_VSA4	fr_alkyl_carbamate	fr_phenol	

Table 4: List of chemical features used

BertzCT	PEOE_VSA14	SlogP_VSA1
EState_VSA2	PEOE_VSA3	SlogP_VSA12
EState_VSA3	PEOE_VSA4	SlogP_VSA2
EState_VSA4	PEOE_VSA5	SlogP_VSA3
EState_VSA5	PEOE_VSA7	SlogP_VSA5
EState_VSA6	PEOE_VSA8	SlogP_VSA6
EState_VSA8	PEOE_VSA9	TPSA
EState_VSA9	SMR_VSA10	VSA_EState9
MolMR	SMR_VSA3	fr_allylic_oxid
PEOE_VSA1	SMR_VSA7	fr_bicyclic
PEOE_VSA11	SMR_VSA9	fr_pyridine

Table 5: Features selected by LASSO at $\alpha = 0.01$ level.