CS181, Practical № 1

Team randomJungles: Nicole Lee, Harry McNamara, Luke Mueller 02/12/2016

Feature analysis and dimension reduction

Our initial approach to feature engineering and data pre-processing involved looking at the given 256 binary features given by RDKit to see if we could exploit sparsity in the data to reduce the dimensionality of our problem.

We first imported the entire 'train.csv' data set, and constructed the covariance matrix. From inspection it was clear that the features were very sparse.

A simple summation over columns revealed that all but 31 of the columns were identically zero, and thus uninformative to training our models. We discarded this data, and the resulting covariance matrix was significantly less sparse.

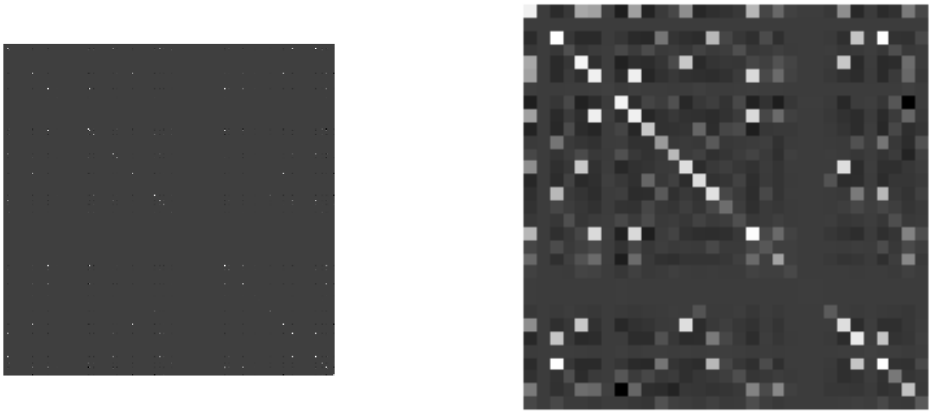

Figure 1: (left) The covariance matrix reveals the sparsity of the initial feature set. (right) The reduced feature set is equally informative

We then prodded a bit more for redundancy in the data by doing singular value decomposition (SVD). In SVD, you can decompose a matrix A with dimension $m \times n$ as:

$$A = U\Sigma V^T$$

Where U is an $m \times m$ dimensional matrix of left singular vectors, Σ is an $m \times n$ rectangular-diagonal matrix of singular values, and V is the matrix of right singular vectors.

Performing this decomposition revealed that, even with null columns eliminated, there were three singular dimensions (i.e. our reduced feature matrix has rank $r = 28 < n = 31$). After poking around, we were able to reduce our dimensionality further (e.g. by removing one feature that was totally redundant with another) such that SVD was full-rank.

With the data thus reduced, we used PCA to rotate the data to it's principle axes (i.e., such that the covariance matrix was diagonal). This allows us to experiment with reducing dimensionality further, by representing a majority of the variance within the features using a compressed feature representation.

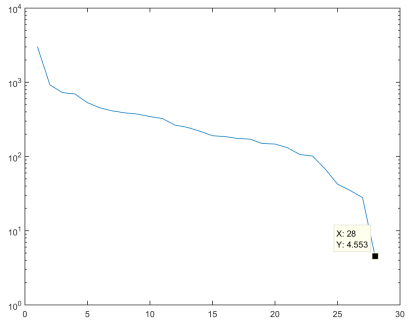

Figure 2: After reduction, there are still 3 singular dimensions

We then tested both the reduced and the rotated representations of the data on the distribution sklearn linear regression and random forests. We found that they performed identically for all, yielding RMS errors (when trained on a 900k subset and validated on a 100k subset) of 0.298 (for LR) and 0.272 (for RF). Moreover, a subset of only the first 10 principal components yielded a nearly equal RMS for linear regression (0.313) and, when baked into a degree 4 polynomial regression, outperformed the minimal linear regression threshold (with RMS = 0.275). Performing similar order regressions on the full or even sparsity-reduced (rank 28) dataset was much more computationally intensive.

Given that the results of the regression seemed relatively insensitive to pruning of sparsity and rotating to principle axes, we thought a good strategy to pursue would be to use RDKit to bake even more features into the feature matrix, then use dimension reduction to make our feature matrix computationally tractable.

Random Forests Hyperparameter Tuning

Random Forest (RF) regression was a follow-up solution to tackling these data, given its size, sparsity, and potential complexity. Previous analyses of RF models have shown them to be consistent and capable of adapting to sparsity in the sense that rate of convergence depends only on the number of strong features in the model and not on how many noise variables are present (Biau, G. Analysis of a Random Forests Model. (2012). Journal of Machine Learning Research. 1063-1095).

We opted to use the RandomForestRegressor function from the Python package sci-kit learn as we were confident in its ability to handle the size of our data, and because our team members were familiar with ensemble methods in Python.

In initial testing, we ran the RF function on our data using a variety of hyper-parameters, hoping to tune them to the extent that fit (RMSE) was improved over the defaults set by sci-kit learn. Using sci-kit learn's gridsearchCV we iterated over different values for $n\_estimators$, and $max\_features$, as suggested in the documentation. Increasing $n\_estimators$ increased the number of decision trees in the forest, while increasing $max\_features$ increased the size of random subsets of features considered when splitting a node. In both cases, marginal increases in fit were seen at the expense of significant computation time (see table below). Indeed, we quickly realized that changes in mean validation score (MVC) between models of different hyper-parameters were trivial (the optimal model simply maximized $n\_estimators$ and $max\_features$), and despite RF's ability to overcome sparsity, model fit would only be improved by altering the original set of features.

Model	$n\_estimators$	$max\_features$	MVC
default	10	auto	0.55140
1	5	auto	0.55123
2	10	sqrt	0.55144
3	15	log2	0.55151
4	5	10	0.55123
5	10	25	0.55141
6	15	50	0.55153

RDKit Feature Extraction

Because of the limited improvements we were seeing in RMSE using our regression methods, we decided to use RDKit to extract additional features to improve our predictions. The AllChem module within the RDKit library allowed us to select between different molecular fingerprints, radii, and bit sizes. With limited testing (due to incredible computation time and memory requirement), we decided to use feature-based circular fingerprints (i.e., Morgan/Circular Fingerprint). The Morgan fingerprints were chosen because of their relative newness and completeness of features. Not only does it extract information about the atoms in the smile, but it also gathers information about the bonds and connectivity between atoms in the molecule and the chemical traits of the molecule (i.e., basic/acidic, donor/acceptor etc.). This package allows for different radii of feature extraction, which represents the number of bonds to take into account in the neighborhood of each atom. Typical radius choices range from 0-3, but in our case, we chose a radius of 2 in order to gather maximal information without slowing down our analysis because of too much data. Using these parameters for feature extraction yielded a total of 2048 features for each molecule, and after running a simple RF model under default parameters we achieved an RMSE of 0.07 against validation data. Satisfied with this result, we performed no further experimentation with RDKit.

Dimensionality reduction of the higher dimensional dataset

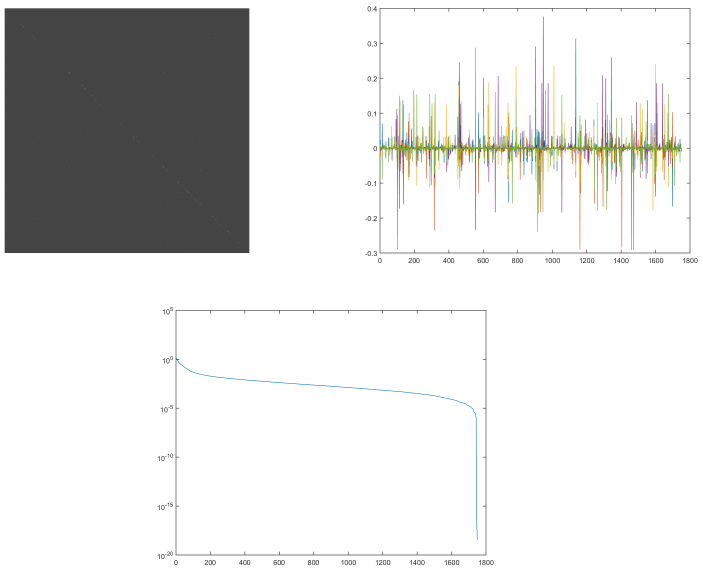

Figure 3: (top left) Covariance matrix of the enriched feature matrix. (top right) 5 largest principal components. (bottom) eigenvalue spectrum associated with principal components.

The richer, 2048-feature data set generated with RDKit presented substantial computational challenges, so in parallel to running the full feature set through a random forest, we also sought to do PCA to capture most of the variance in the feature set with a smaller dimensionality.

Removing null features yielded 1751 feature columns. PCA on this data set revealed that it was substantially richer than the previous set: the matrix was close to full rank, and its largest principal nearly spanned the full feature space.

By considering the cumulative eigenvalue distribution, we found that we could capture approximate 75% of the variance with the 100 largest components, 85% of the variance with 200 components, and 95% with 500.

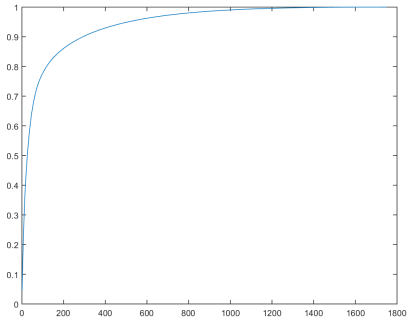

Figure 4: Cumulative distribution of eigenvalues

While in the process of training on the full feature set, we experimented with training the first 100 principal components on a 900k subset of the training data, and then validating on the remaining 100k just in case our memory capacities were not sufficient. We found that a 100 feature random forest took approximately one hour to run, and returned RMSE of 0.135 - not as good as the full set, but much better than the original 256 dimensional set! Moreover a simple linear regression (i.e. using a model $y_n^{pred} = w^T x_n$) ran almost instantly, and returned RMSE of 0.184.

Prediction using the full feature set

Despite our earlier doubts that training on the new feature set from RDKit would work, after a few hours of dead kernels and some compromises, we finally obtained predictions on the test set using most of the full 2048 features. The compromises we had to make were as follows: (1) training on the full 1 million molecules was proving impossible, so we shuffled the rows and subsetted 500K to train upon. Additionally, because of the sparsity of some of the features, we weeded out uninformative features by simply including features that were included in > 100 molecules. This reduced the number of features from 2048 to 1585. We fit our model using the basic RandomForestRegressor in python and predicted on our test set in two batches to reduce memory usage. The batches were stitched together, then submitted to Kaggle as our final output. This submission yielded an RMSE of 0.07435, our best result yet.

SVD and Feature Engineering

In [2]:
```
%matplotlib inline

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn import linear_model
from sklearn.ensemble import RandomForestRegressor

from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
```

In [3]:
```
"""
Read in train and test as Pandas DataFrames
"""
#df_raw = pd.read_csv("train.csv")# df_test = pd.read_csv("test.csv")
df_raw = pd.read_csv("Train100.csv")

# shuffle data set
df_shuff = df_raw.iloc[np.random.permutation(len(df_raw))]
```

In [4]:
```
df_raw.head()
```

Out[4]:

	1	2	3	4	5	6	7	8	9	10	...	92	93	94	95	96	97	98	99	100	gap
0	0.671980	-2.74760	-0.90490	-1.37640	-0.327940	0.65507	0.48472	-0.23502	-1.45630	0.018838	...	-0.36252	0.23768	0.212940	0.67475	-0.249000	-0.233140	0.33807	0.70074	0.21382	1.19
1	-0.077896	0.27362	2.45170	-0.35339	0.095019	-1.02340	0.53486	1.32690	0.77190	0.599490	...	-0.10172	-0.32479	0.318260	0.25401	0.259080	0.172780	-0.13304	0.79112	0.18406	1.60
2	1.280300	-1.52000	0.94333	1.10460	1.274400	0.82299	0.68534	1.19940	-0.32390	-0.039913	...	-0.62211	0.82954	-0.354570	-0.78441	-0.014769	-0.033582	-0.12543	0.47659	0.12949	1.49
3	-0.936130	1.61680	1.25940	-0.66870	0.615760	0.60529	1.71630	-0.87249	-0.24715	0.503820	...	0.42405	-0.12964	-0.303140	0.39505	-0.265540	-0.318840	0.69027	0.10446	0.18172	1.36
4	-2.007900	-1.61490	-0.98113	-0.74726	1.394100	-0.19481	0.32409	0.79098	0.50147	-0.156480	...	-0.39674	0.61034	0.029535	-0.14028	-0.169010	-0.559720	0.35737	0.64763	0.22416	1.98

5 rows × 101 columns

In [ ]:
```
##  convert to numpy array for statistical analysis
# trainMat = df_train.as_matrix()
#print trainMat.shape
#trainMat = trainMat.T[2:]
# trainMat = trainMat.T[1:].T

# print trainMat
```

In [3]:
```
# find covariance matrix for PCA
# CovMat = np.cov(TrainParams.T)
# print CovMat
# FIND PRINCIPLE COMPONENTS
# l, v = np.linalg.eig[CovMat]


# takes too long - do this part in MATLAB, then do a csv read to re-load rota
ted data
df_red = pd.read_csv("train_red.csv")
df_rot = pd.read_csv("train_rot.csv")
# shuffle data set
df_rot_shuff = df_rot.iloc[np.random.permutation(len(df_rot))]
```

In [5]:
```
#10 fold cross-validation to test data eventually?
# for now, just get 10% of data set for validation?

#df_test = df_raw[:100000]
#df_train = df_raw[100001:]

df_test = df_shuff[:100000]
df_train = df_shuff[100001:]

df_test.head()
```

Out[5]:

	1	2	3	4	5	6	7	8	9	10	...	92	93	94	95	96	97	98	99	100	gap
180251	0.72714	-0.47628	0.68548	0.607260	0.83073	0.83008	-1.507500	1.906400	0.852240	-0.004386	...	-0.28545	0.156300	-0.210000	0.407660	-0.315430	0.118210	0.41137	0.16682	0.059871	2.16
796873	-0.18371	1.12140	-1.39890	0.752900	2.34060	0.05680	0.350770	-0.334390	-0.402100	1.053100	...	-0.38779	0.447970	-0.001468	0.044464	0.001598	0.086369	0.25111	0.41153	0.571910	1.86
268276	-0.15060	-2.19190	0.88661	1.415700	0.43688	-2.09430	0.323340	-0.831040	0.034224	-0.876590	...	-0.23265	0.458320	0.465790	0.627030	-0.222890	-0.005847	0.56771	0.56310	-0.055066	1.29
545543	1.07760	-1.40470	-0.28090	-0.074416	1.47520	-0.38545	3.118600	1.477300	0.189690	-0.583470	...	0.19391	0.085452	0.252270	-0.472650	-0.549060	-0.368150	0.35092	0.36934	0.445850	1.60
423666	0.97558	-0.68237	-1.60810	-0.942540	0.05695	0.19968	-0.016015	0.081247	0.808870	-1.898000	...	-0.32485	0.292500	0.005777	0.345000	-0.294590	-0.048540	0.57451	0.45096	0.321610	2.11

5 rows × 101 columns

In [5]:
```
df_red_test = df_red[:100000]
df_red_train = df_red[100001:]

df_rot_test = df_rot[:100000]
df_rot_train = df_rot[100001:]

#df_rot_test = df_rot_shuff[:100000]
#df_rot_train = df_rot_shuff[100001:]

df_rot_train.head()
```

Out[5]:

	feat_001	feat_002	feat_003	feat_004	feat_005	feat_006	feat_007	feat_008	feat_009	feat_010	...	feat_020	feat_021	feat_022	feat_023	feat_024	feat_025	feat_026	feat_027	feat_028	gap
100001	-0.31780	1.06390	-0.15708	0.26591	-1.83440	-0.11096	0.80133	-0.65678	0.80561	-1.514500	...	0.244900	-0.109890	-0.097408	0.30806	0.92627	-0.047037	0.004986	0.001478	-0.99612	1.90
100002	0.67780	1.23510	-0.14180	0.77887	-1.01320	-1.00000	0.86199	-0.60251	1.08020	0.077677	...	0.175590	-0.077588	-0.126100	0.31434	0.92842	-0.051463	0.005864	0.001529	-0.99612	2.64
100003	0.33303	-0.24289	0.45346	0.55696	-0.55435	-1.50610	0.22528	-0.30043	0.54306	-0.324070	...	0.190670	-0.026750	-0.054827	0.29763	0.92728	-0.048534	0.004481	0.001410	-0.99619	1.74
100004	-1.37350	0.04203	0.89451	1.06970	-1.82500	-0.81984	0.69116	-0.39065	0.81566	-0.176780	...	0.095164	-0.022447	-0.096790	0.32140	0.92914	-0.048473	0.003036	0.001483	-0.99612	2.07
100005	-0.32837	-0.70804	-0.94477	1.25320	-1.03430	-1.21020	0.67593	-0.54604	1.00700	-0.299720	...	0.078359	-0.064004	-0.143430	0.29631	0.93252	-0.051648	0.002499	0.001380	-0.99617	1.80

5 rows × 29 columns

In [6]:
```
#store gap values
Y_train = df_train.gap.values
Y_test = df_test.gap.values

Y_red_train = df_red_train.gap.values
Y_red_test = df_red_test.gap.values

Y_rot_train = df_rot_train.gap.values
Y_rot_test = df_rot_test.gap.values
#row where testing examples start
test_idx = df_train.shape[0]
#delete 'Id' column
#df_test = df_test.drop(['Id'], axis=1)
#delete 'gap' column
df_train = df_train.drop(['gap'], axis=1)
df_test = df_test.drop(['gap'], axis=1)

df_red_train = df_red_train.drop(['gap'], axis=1)
df_red_test = df_red_test.drop(['gap'], axis=1)

df_rot_train = df_rot_train.drop(['gap'], axis=1)
df_rot_test = df_rot_test.drop(['gap'], axis=1)
```

In [19]:
```
#DataFrame with all train and test examples so we can more easily apply featu
re engineering on
df_all = pd.concat((df_train, df_test), axis=0)
df_all.head()

df_rot_all = pd.concat((df_rot_train, df_rot_test), axis=0)
df_rot_all.head()
```

Out[19]:

	feat_001	feat_002	feat_003	feat_004	feat_005	feat_006	feat_007	feat_008	feat_009	feat_010	...	feat_019	feat_020	feat_021	feat_022	feat_023	feat_024	feat_025	feat_026	feat_027	feat_028
434949	0.898670	-1.11090	0.18956	-0.35985	-0.25562	0.071154	-0.77098	0.96908	-0.59665	-0.059767	...	0.613870	-0.23150	0.019500	-0.059335	-0.30408	-0.93290	0.006876	-0.045498	0.001382	-0.99612
308772	-0.453910	-2.02260	-0.36717	-1.54940	-0.66758	0.469400	-1.02840	0.88055	-0.39145	0.032931	...	0.630600	-0.13128	-0.020743	-0.096501	-0.31017	-0.92819	0.002519	-0.047293	0.001447	-0.99611
30252	-1.324300	-0.65704	-0.61896	-0.49285	-1.79070	0.003982	-0.45323	0.59592	-0.96008	-0.044186	...	0.997270	-0.33868	0.004312	-0.042667	-0.34114	-0.92408	0.004917	-0.040895	0.001680	-0.99614
757418	0.027234	-1.44340	2.07950	-0.60351	-0.59694	-0.313590	-0.27555	-0.39128	-1.01790	0.150750	...	0.097643	0.29208	-0.032956	-0.069577	-0.28419	-0.93401	0.003712	-0.053198	0.000786	-0.99668
712288	0.183920	0.25354	-0.17156	-1.03250	-0.10872	0.416340	-0.95407	0.15638	-0.61046	-0.133210	...	0.660860	-0.21747	0.006252	-0.115380	-0.30672	-0.93062	0.004532	-0.052651	0.001379	-0.99613

5 rows × 28 columns

In [45]:
```
"""
Example Feature Engineering

this calculates the length of each smile string and adds a feature column wit
h those lengths
Note: this is NOT a good feature and will result in a lower score!
"""
#smiles_len = np.vstack(df_all.smiles.astype(str).apply(lambda x: len(x)))
#df_all['smiles_len'] = pd.DataFrame(smiles_len)
```

In [9]:
```
X_train = df_train.values
X_test = df_test.values
```

In [7]:
```
#Drop the 'smiles' column
#df_all = df_all.drop(['smiles'], axis=1)
#vals = df_all.values
#X_train = vals[:test_idx]
#X_test = vals[test_idx:]

# drop smiles
df_test = df_test.drop(['smiles'], axis=1)
df_train = df_train.drop(['smiles'], axis=1)

X_train = df_train.values
X_test = df_test.values

X_red_train = df_red_train.values
X_red_test = df_red_test.values

X_rot_train = df_rot_train.values
X_rot_test = df_rot_test.values

print "Train features:", X_train.shape
print "Train gap:", Y_train.shape
print "Test features:", X_test.shape
```

```
Train features: (899999L, 256L)
Train gap: (899999L,)
Test features: (100000L, 256L)
```

In [8]:
```
print X_rot_train.shape
print Y_rot_train.shape
print X_rot_test.shape

#print X_rot_train
```

```
(899999L, 28L)
(899999L,)
(100000L, 28L)
```

In [8]:
```
idx = np.array([0, 2,3,4, 6, 7,8,9,10,11,12,13,14,15,16,17,18,19, 21, 22, 24,
25,26,27])

print X_train.T[idx].T.shape, X_rot_train.shape

#X_comb_train = X_red_train.T[idx].T
#X_comb_test = X_red_test.T[idx].T
X_comb_train = X_rot_train.T[:5].T
X_comb_test = X_rot_test.T[:5].T


#X_comb_train = np.concatenate((X_train, X_rot_train), axis=1)
print X_comb_train.shape
#X_comb_test = np.concatenate((X_test,X_rot_test), axis=1)
print X_comb_test.shape
```

```
(899999L, 24L) (899999L, 28L)
(899999L, 10L)
(100000L, 10L)
```

In [17]:
```
LR = LinearRegression()
LR.fit(X_train, Y_train)
LR_pred = LR.predict(X_test)

LR_ErrorVec = np.array(Y_test)-np.array(LR_pred)
LR_RMSErrorScore = np.sqrt(0.00001*np.sum(np.square(LR_ErrorVec)))
```

```
  File "<ipython-input-17-817cced45e4c>", line 7

    ^
SyntaxError: invalid syntax
```

In [9]:
```
LR_red = LinearRegression()
LR_red.fit(X_red_train,Y_red_train)
LR_red_pred = LR_red.predict(X_red_test)

LR_rot = LinearRegression()
LR_rot.fit(X_rot_train, Y_rot_train)
LR_rot_pred = LR_rot.predict(X_rot_test)
```

In [19]:
```
LR_comb = LinearRegression()
LR_comb.fit(X_comb_train, Y_rot_train)
LR_comb_pred = LR_comb.predict(X_comb_test)
```

In [19]:
```
LR_ErrorVec = np.array(Y_test)-np.array(LR_pred)
LR_RMSErrorScore = np.sqrt(0.00001*np.sum(np.square(LR_ErrorVec)))

#LR_ErrorVec_red = np.array(Y_red_test)-np.array(LR_red_pred)
#LR_RMSErrorScore_red = np.sqrt(0.00001*np.sum(np.square(LR_ErrorVec_red)))

#print ErrorVec
print "Normal: ", LR_RMSErrorScore
#print "Reduced: ", LR_RMSErrorScore_red
```

```
Normal:  0.184470251717
```

In [16]:
```
LR_ErrorVec_rot = np.array(Y_rot_test)-np.array(LR_rot_pred)
LR_RMSErrorScore_rot = np.sqrt(0.00001*np.sum(np.square(LR_ErrorVec_rot)))

LR_ErrorVec_comb = np.array(Y_rot_test)-np.array(LR_comb_pred)
LR_RMSErrorScore_comb = np.sqrt(0.00001*np.sum(np.square(LR_ErrorVec_comb)))

#print ErrorVec
print "Rotated: ", LR_RMSErrorScore_rot
print "low variance removed: ", LR_RMSErrorScore_comb
```

```
Rotated:  0.298185020708
low variance removed:  0.31385967039
```

In [24]:
```
degree = 4

PolyN = make_pipeline(PolynomialFeatures(degree), Ridge())
PolyN.fit(X_train.T[:10].T, Y_train)
Y_pred_PolyN = PolyN.predict(X_test.T[:10].T)
```

In [23]:
```
PolyN_ErrorVec_red = np.array(Y_test)-np.array(Y_pred_PolyN)
PolyN_RMSErrorScore_red = np.sqrt(0.00001*np.sum(np.square(PolyN_ErrorVec_red
)))
print "polynomial: ", PolyN_RMSErrorScore_red
```

```
polynomial:  0.257640834212
```

In [10]:
```
RF = RandomForestRegressor(n_estimators = 10)
RF.fit(X_train, Y_train)
RF_pred = RF.predict(X_test)
```

In [20]:
```
RF_red = RandomForestRegressor(n_estimators = 10)
RF_red.fit(X_red_train, Y_red_train)
RF_red_pred = RF_red.predict(X_red_test)
```

In [21]:
```
RF_rot = RandomForestRegressor(n_estimators = 10)
RF_rot.fit(X_rot_train, Y_rot_train)
RF_rot_pred = RF_rot.predict(X_rot_test)
```

In [22]:
```
RF_comb = RandomForestRegressor()
RF_comb.fit(X_comb_train, Y_train)
RF_comb_pred = RF_comb.predict(X_comb_test)
```

In [11]:
```
ErrorVec = np.array(Y_test)-np.array(RF_pred)
RMSErrorScore = np.sqrt(0.00001*np.sum(np.square(ErrorVec)))
print "Normal: ", RMSErrorScore
```

```
Normal:  0.135494705044
```

In [23]:
```
ErrorVec = np.array(Y_test)-np.array(RF_pred)
RMSErrorScore = np.sqrt(0.00001*np.sum(np.square(ErrorVec)))

ErrorVec_red = np.array(Y_red_test)-np.array(RF_red_pred)
RMSErrorScore_red = np.sqrt(0.00001*np.sum(np.square(ErrorVec_red)))

ErrorVec_rot = np.array(Y_rot_test)-np.array(RF_rot_pred)
RMSErrorScore_rot = np.sqrt(0.00001*np.sum(np.square(ErrorVec_rot)))

ErrorVec_comb = np.array(Y_test)-np.array(RF_comb_pred)
RMSErrorScore_comb = np.sqrt(0.00001*np.sum(np.square(ErrorVec_comb)))
#print ErrorVec
print "Normal: ", RMSErrorScore
print "Reduced: ", RMSErrorScore_red

print "Rotated: ", RMSErrorScore_rot
print "Combined: ", RMSErrorScore_comb
```

```
Normal:  0.271792759332
Reduced:  0.271796080426
Rotated:  0.271778751195
Combined:  0.271843611671
```

In [19]:
```
print RF_rot.feature_importances_
```

```
[ 0.00936447  0.02869281  0.2621802   0.01818699  0.30486678  0.0088479
  0.16722489  0.00284595  0.01039439  0.00975328  0.0033085   0.00703939
  0.00521937  0.00553985  0.01871409  0.02350245  0.01911731  0.00315933
  0.01089921  0.00312364  0.00699746  0.02044814  0.0058451   0.01898798
  0.0054651   0.01265346  0.00307515  0.00454682  0.          0.          0.
]
```

In [39]:
```
X_train_red = X_train.T[np.array([2, 4, 6])].T
X_test_red = X_test.T[np.array([2, 4, 6])].T
print X_train_red.shape

RF_red = RandomForestRegressor(n_estimators = 10)
RF_red.fit(X_train_red, Y_train)
RF_red_pred = RF_red.predict(X_test_red)
```

```
(899996L, 3L)
```

In [42]:
```
ErrorVec_red = np.array(Y_test)-np.array(RF_red_pred)
RMSErrorScore_red = np.sum(np.sqrt(np.square(ErrorVec_red)))
print "Reduced: ", RMSErrorScore_red

print RF_red.feature_importances_
```

```
Reduced:  29469.5036036
[ 0.0203553   0.97863445  0.00101025]
```

In [20]:
```
#write_to_file('errorvec.csv', ErrorVec)
#write_to_file('errorvec_rot.csv', ErrorVec_rot)
#write_to_file('testdata.csv', X_test)
#write_to_file('testdata_rot.csv',X_rot_test)
```

In [21]:
```
def write_to_file(filename, predictions):
    with open(filename, "w") as f:
        f.write("Id,Prediction\n")
        for i,p in enumerate(predictions):
            f.write(str(i+1) + "," + str(p) + "\n")
```

In [22]:
```
write_to_file("errorvec.csv", ErrorVec)
write_to_file("errorvec_rot.csv", ErrorVec_rot)

write_to_file("feature_importances.csv", RF.feature_importances_)
write_to_file("rot_feature_importances.csv", RF_rot.feature_importances_)
```

In [48]:
```
write_to_file("sample1.csv", LR_pred)
write_to_file("sample2.csv", RF_pred)
```

Matlab Code for SVD analysis

```
cd C:\Users\Harry\Desktop\Spring' 2016'\CS181\Practical1\

data = csvread('New_Features.csv',1,1);
params = data(:,1:end-1);
gaps = data(:,end); % get data structres

% truncate by eliminating empty columns
idx = find(sum(params)>0); % indices of populated columns
params_red = params(:,idx);

% perform SVD, but only on subset due to large memorry requirements (e.g.
% just use 10k)

[U,S,V] = svd(params_red,0);

semilogy(sum(S));
clf;

rho = 28; % from inspection of signular values
%also can see from rank(params_red)
params_red2 = params_red(:,[1:22 24:end]);
params_red3 = params_red2(:,[1:2 4:end]);
params_redT = params_red3(:,[1:22 24:end]);

rank(params_redT)
data_red = params_redT; data_red(:,end+1) = gaps;
csvwrite('train_red.csv',data_red);

% rotate to principle components
CMat = cov(params_redT);
[V,D] = eig(CMat);
figure(); plot(diag(D));
```

```
% rearrange to principle axes
V = V(:,end:-1:1);
params_rot = params_redT*V;
data_rot = params_rot; data_rot(:,end+1) = gaps;
csvwrite('train_rot.csv',data_rot);
```

```
Error using csvread (line 34)
File not found.

Error in SVD_analysis (line 3)
data = csvread('New_Features.csv',1,1);
```

Published with MATLAB® R2015a

RANDOM FOREST HYPERPARAMETER TUNING:

```
%matplotlib inline
import scipy as sp
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn import preprocessing
from sklearn import grid_search
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import mean_squared_error

df_train = pd.read_csv("train.csv")
df_test = pd.read_csv("test.csv")

df_train['is_train'] = np.random.uniform(0, 1, len(df_train)) <= .75train,
validate = df_train[df_train['is_train']==True],
df_train[df_train['is_train']==False]

#store gap values
y_train = train.gap.values
y_validate = validate.gap.values
#delete 'gap','is_train', smiles' columns
train = train.drop(['gap'], axis=1)
train = train.drop(['is_train'], axis=1)
train = train.drop(['smiles'], axis=1)
validate = validate.drop(['gap'], axis=1)
validate = validate.drop(['is_train'], axis=1)
validate = validate.drop(['smiles'], axis=1)
x_train = train.values
x_validate = validate.values

params = {'max_features':('auto','sqrt','log2',10,25,50),
'n_estimators':[5,10,15]}

%%time
rf = RandomForestRegressor()
rf_search = GridSearchCV(estimator = rf, param_grid = params, cv = 5)
rf_search.fit(x_train,y_train)
rf_search.grid_scores_


params2 = {'max_features':('auto','sqrt','log2',10,15,20,25,30,35,40,45,50)}

%%time
rf = RandomForestRegressor()
rf2_search = GridSearchCV(estimator = rf, param_grid = params2, cv = 5)
rf2_search.fit(x_train,y_train)
rf2_search.grid_scores_
best = rf2_search.best_estimator_

plt.hist(y_validate,color='r',alpha=0.4,bins=75)
plt.hist(best_predictions,color='b',alpha=0.4,bins=75)
#plt.legend()
plt.show()
```

```
resids = best_predictions - y_validate
plt.scatter(best_predictions,resids)
plt.show()

importances = best.feature_importances_
std = np.std([tree.feature_importances_ for tree in best.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(x_train.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f],
importances[indices[f]]))

# Plot the feature importances of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(range(x_train.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(x_train.shape[1]), indices)
plt.xlim([-1, x_train.shape[1]])
plt.show()


indices[:31]
x_train_sub = x_train[:,indices[:31]]

x_validate_sub = x_validate[:,indices[:31]]

%%time
rf_sub_model =  RandomForestRegressor()
rf_sub_model.fit(x_train_sub,y_train)
sub_predictions = rf_sub_model.predict(x_validate_sub)

%%time
num_features = [10,15,20]
for i in num_features:
    x_train_sim = x_train[:,indices[:i]]
    x_validate_sim = x_validate[:,indices[:i]]
    rf_sim =  RandomForestRegressor()
    rf_sim.fit(x_train_sim,y_train)
    predicted = rf_sim.predict(x_validate_sim)
    print("RMSE: %f" % np.sqrt(mean_squared_error(y_validate,predicted)))

plt.hist(y_validate,color='r',alpha=0.4,bins=25)
plt.hist(sub_predictions,color='b',alpha=0.4,bins=25)
#plt.legend()
plt.show()


#xgboost
import pickle
import xgboost as xgb
import numpy as np
```

```
from sklearn.cross_validation import KFold, train_test_split
from sklearn.metrics import confusion_matrix, mean_squared_error
from sklearn.grid_search import GridSearchCV


rng = np.random.RandomState(31337)
%%time
kf = KFold(y_train.shape[0], n_folds=2, shuffle=True, random_state=rng)
for train_index, test_index in kf:
    xgb_model = xgb.XGBRegressor().fit(x_train,y_train)
    predictions = xgb_model.predict(x_validate)
    actuals = y_validate
    print(mean_squared_error(actuals, predictions))
```

Extraction of Additional Features Using RDKit

```
from rdkit import Chem, DataStructs
from rdkit.Chem import AllChem
import pandas as pd
import numpy as np


df_train = pd.read_csv("train.csv")
df_test = pd.read_csv("test.csv")
```

Obtain new features for training set

```
smiles_train = df_train['smiles']


mols=[]
for smile in smiles_train:
    m = Chem.MolFromSmiles(smile)
    mols.append(m)


fps = [AllChem.GetMorganFingerprintAsBitVect(m, 2) for m in mols]


np_fps = []
for fp in fps:
    arr = np.zeros((1,))
    DataStructs.ConvertToNumpyArray(fp, arr)
    np_fps.append(arr)


len(np_fps)



1000000



len(np_fps[0])



2048



matrix = np.vstack(np_fps)
```

```
matrix.shape
```

```
(1000000, 2048)
```

```
df_new = pd.DataFrame(matrix)
```

```
df_new.head()
```

	0	1	2	3	4	5	6	7	8	9	...	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 2048 columns

```
df_new['gap'] = df_train['gap']
```

```
df_new['smiles'] = df_train['smiles']
```

```
df_new.to_csv('New_Features.csv')
```

New features for test dataset

```
smiles_test = df_test['smiles']
```

```
mols_test=[]
for smile in smiles_test:
    m = Chem.MolFromSmiles(smile)
    mols_test.append(m)
```

```
fps_test = [AllChem.GetMorganFingerprintAsBitVect(m, 2) for m in mols_test]
```

```
np_fps_test = []
for fp in fps_test:
    arr = np.zeros((1,))
    DataStructs.ConvertToNumpyArray(fp, arr)
    np_fps_test.append(arr)
```

```
matrix_test = np.vstack(np_fps_test)
df_new_test = pd.DataFrame(matrix_test)

df_new_test['smiles'] = df_test['smiles']


df_new_test.to_csv('New_Features_test.csv')
```

Random Forest on Expanded Features

```
import pandas as pd
import numpy as np
from sklearn import linear_model as lm
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error


df_train_new = pd.read_csv("New_Features.csv")
```

Is there improvement in RMSE using the training data?

```
train_shuff = df_train_new.iloc[np.random.permutation(len(df_train_new))]


train_shuff = train_shuff.drop(['smiles'], axis=1)
sh_train = train_shuff[:500000]
sh_test = train_shuff[500001:]


sh_X_train = sh_train.drop(['gap'], axis=1)
sh_Y_train = sh_train.gap.values
sh_X_test = sh_test.drop(['gap'], axis=1)
sh_Y_test = sh_test.gap.values


rf = RandomForestRegressor()
rf.fit(sh_X_train, sh_Y_train)
rf_pred = rf.predict(sh_X_test)


print "Random Forest RMSE with new features"
np.sqrt(mean_squared_error(sh_Y_test, rf_pred))

Random Forest RMSE with new features




0.07647029959019018
```

Here, we partitioned the training dataset so that we could find out if this method of using newly extracted features and random forest would improve our RMSE. We see that there has been a large improvement in RMSE, going from approximately 0.27 to 0.076. So, it seems promising to submit this to the kaggle by predicting on the actual test dataset.

Do the same on actual test set

We're getting a lot of dead kernels when working with the full training dataset. Will be partitioning

as before to predict on the test dataset.

```
train_shuff = df_train_new.iloc[np.random.permutation(len(df_train_new))]

train_shuff = train_shuff.drop(['smiles'], axis=1)

sh_train = train_shuff[:500000]

X_train = sh_train.drop(['gap'], axis=1)

X_train = X_train.drop(['Unnamed: 0'], axis=1)

y_train = sh_train.gap.values
```

Our data is too large to run random forest. this filters out sparse features

```
good_cols = np.nonzero(X_train.sum(axis=0) > 100)

np.save("good_cols", good_cols)

X_train = X_train.values

X_train_small = X_train[:,good_cols[0]]

X_train_small.shape



(500000, 1585)



rf = RandomForestRegressor()
rf.fit(X_train_small, y_train)



RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
           verbose=0, warm_start=False)
```

```
from sklearn.externals import joblib
joblib.dump(rf, 'rf.pkl')
```

```
['rf.pkl',
 'rf.pkl_01.npy',
 'rf.pkl_02.npy',
 'rf.pkl_03.npy',
 'rf.pkl_04.npy',
 'rf.pkl_05.npy',
 'rf.pkl_06.npy',
 'rf.pkl_07.npy',
 'rf.pkl_08.npy',
 'rf.pkl_09.npy',
 'rf.pkl_10.npy',
 'rf.pkl_11.npy',
 'rf.pkl_12.npy',
 'rf.pkl_13.npy',
 'rf.pkl_14.npy',
 'rf.pkl_15.npy',
 'rf.pkl_16.npy',
 'rf.pkl_17.npy',
 'rf.pkl_18.npy',
 'rf.pkl_19.npy',
 'rf.pkl_20.npy',
 'rf.pkl_21.npy',
 'rf.pkl_22.npy',
 'rf.pkl_23.npy',
 'rf.pkl_24.npy',
 'rf.pkl_25.npy',
 'rf.pkl_26.npy',
 'rf.pkl_27.npy',
 'rf.pkl_28.npy',
 'rf.pkl_29.npy',
 'rf.pkl_30.npy',
 'rf.pkl_31.npy',
 'rf.pkl_32.npy',
 'rf.pkl_33.npy',
 'rf.pkl_34.npy',
 'rf.pkl_35.npy',
 'rf.pkl_36.npy',
 'rf.pkl_37.npy',
 'rf.pkl_38.npy',
 'rf.pkl_39.npy',
 'rf.pkl_40.npy']
```

Read in test dataset

```
df_test_new = pd.read_csv("New_Features_test.csv")


from sklearn.externals import joblib
rf = joblib.load('rf.pkl')
```

```
good_cols = np.load("good_cols.npy")


df_test1 = df_test_new[:400000]


X_test1 = df_test1.drop(['smiles'], axis=1)


X_test1 = X_test1.drop(['Unnamed: 0'], axis=1)


good_cols = good_cols.tolist()


X_test_small1 = X_test1[good_cols[0]]


rf_pred1 = rf.predict(X_test_small1)


def write_to_file(filename, predictions):
    with open(filename, "w") as f:
        f.write("Id,Prediction\n")
        for i,p in enumerate(predictions):
            f.write(str(i+1) + "," + str(p) + "\n")


df_test2 = df_test_new[400000:]


X_test2 = df_test2.drop(['smiles'], axis=1)


X_test2 = X_test2.drop(['Unnamed: 0'], axis=1)


X_test_small2 = X_test2[good_cols[0]]


rf_pred2 = rf.predict(X_test_small2)


rf_pred = np.hstack((rf_pred1, rf_pred2))


len(rf_pred)
```

```
824230
```

```
write_to_file("rf_with_newfeatures.csv", rf_pred)
```