Data Mining Project:

FINAL REPORT

Nomad2018 Predicting Transparent Conductors

https://www.kaggle.com/c/nomad2018-predict-transparent-conductors/

TEAM MEMBERS:

ARCHIT AWASTHI
RAJ KUMAR

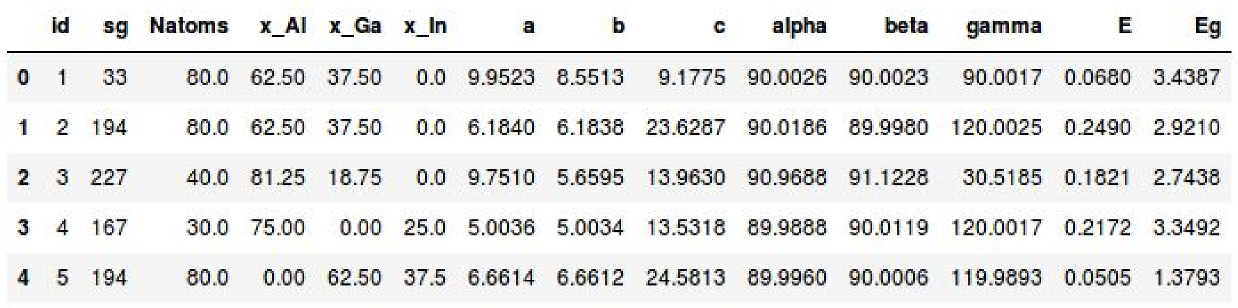
OVERVIEW

The aim of this challenge is to predict the physical properties of specific types of Transparent Conductors.Essentially, these are high-tech materials that can be designed, pretty much atom by atom, from a large range of different quantities of the chemical elements Aluminium (Al), Gallium (Ga), and Indium (In).

There are two targets we need to predict for each material: the *formation energy* (indicator for its stability) and the *bandgap energy* (indicating its transparency). Those will be continuous variables which makes this a regression problem. We are tasked with predicting the “key performance indicators” of each material based on different features.

DATA FORMAT

The data comes in the shape of the familiar two *train* and *test* files (`../input/train.csv` and `../input/test.csv`) together with 1 *geometry.xyz* file per id. This is a small data set with only 3000 materials which means that even with this many files it does not require a lot of memory. In the *train* and *test* sets each row corresponds to a specific material.



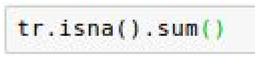
	id	sg	Natoms	x_Al	x_Ga	x_In	a	b	c	alpha	beta	gamma	E	Eg
0	1	33	80.0	62.50	37.50	0.0	9.9523	8.5513	9.1775	90.0026	90.0023	90.0017	0.0680	3.4387
1	2	194	80.0	62.50	37.50	0.0	6.1840	6.1838	23.6287	90.0186	89.9980	120.0025	0.2490	2.9210
2	3	227	40.0	81.25	18.75	0.0	9.7510	5.6595	13.9630	90.9688	91.1228	30.5185	0.1821	2.7438
3	4	167	30.0	75.00	0.00	25.0	5.0036	5.0034	13.5318	89.9888	90.0119	120.0017	0.2172	3.3492
4	5	194	80.0	0.00	62.50	37.5	6.6614	6.6612	24.5813	89.9960	90.0006	119.9893	0.0505	1.3793

- There are 4 composition features: The *percent_atom* for “Al”, “Ga”, and “In” describe the relative composition of the material, i.e. how much of Aluminium, Gallium, and Indium it contains. Those three features all range from 0 to 1. The *number_of_total_atoms* is pretty self explanatory and ranges from 10 to 80; here we already see that only a few values appear to be taken. Those are all numerical features.
- the *spacegroup* column encodes certain symmetry properties of the grid. With exception of the integer *spacegroup* these are floating point variables.
- a, b and c are the lengths of the lattice vectors in angstroms, with values ranging from around 3 to 25 angstroms. Similarly, alpha, beta and gamma are angles in degrees which describe the spatial dimensions and distortion of the grid.
- *formation_energy_ev_atom and bandgap_energy_ev are our two target variables. Those are floating point features describing energies measured in the unit electron volt (eV). We see that they range between zero and about 0.5 (formation) or about 5 (bandgap), respectively.*

DATA PREPROCESSING

MISSING VALUES

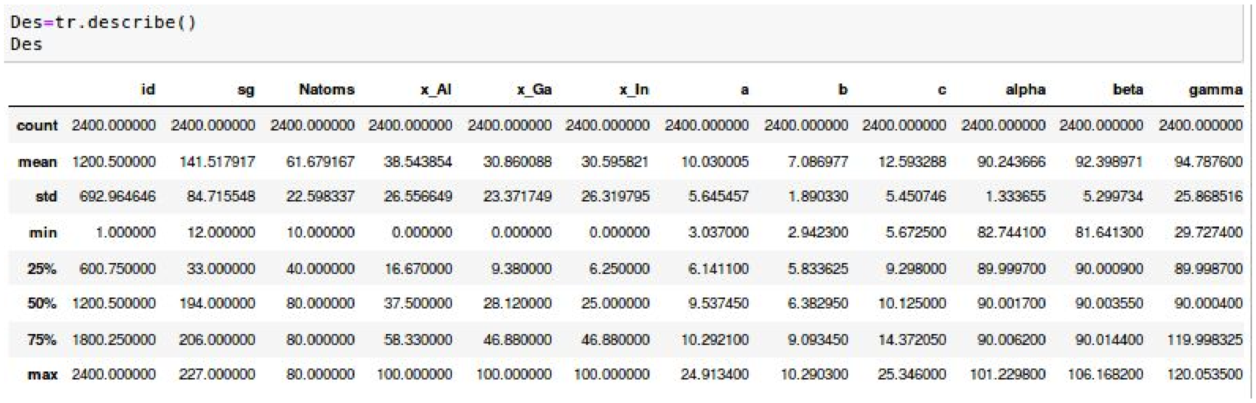
At first , on analysis we found no Missing values were present in our dataset.



```
tr.isna().sum()
```

BASIC STATISTICAL ANALYSIS AND VISUALISATION USING BOXPLOTS

We looked at the basic statistics for each attribute like mean, min, max and the quartiles.



```
Des=tr.describe()
Des
```

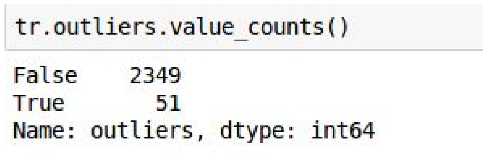
	id	sg	Natoms	x_Al	x_Ga	x_In	a	b	c	alpha	beta	gamma
count	2400.000000	2400.000000	2400.000000	2400.000000	2400.000000	2400.000000	2400.000000	2400.000000	2400.000000	2400.000000	2400.000000	2400.000000
mean	1200.500000	141.517917	61.679167	38.543854	30.860088	30.595821	10.030005	7.086977	12.593288	90.243666	92.398971	94.787600
std	692.964646	84.715548	22.598337	26.556649	23.371749	26.319795	5.645457	1.890330	5.450746	1.333655	5.299734	25.868516
min	1.000000	12.000000	10.000000	0.000000	0.000000	0.000000	3.037000	2.942300	5.672500	82.744100	81.641300	29.727400
25%	600.750000	33.000000	40.000000	16.670000	9.380000	6.250000	6.141100	5.833625	9.298000	89.999700	90.000900	89.998700
50%	1200.500000	194.000000	80.000000	37.500000	28.120000	25.000000	9.537450	6.382950	10.125000	90.001700	90.003550	90.000400
75%	1800.250000	206.000000	80.000000	58.330000	46.880000	46.880000	10.292100	9.093450	14.372050	90.006200	90.014400	119.998325
max	2400.000000	227.000000	80.000000	100.000000	100.000000	100.000000	24.913400	10.290300	25.346000	101.229800	106.168200	120.053500

Then we visualised the above statistics by analysing following boxplots -

- *box-plots of percentage composition of the three elements*
- *box-plots of the three lattice vectors*
- *box-plots of the two target variables*
- *box-plots of the 2 target variables corresponding to the different values of Natoms*
- *box-plots of the 2 target variables corresponding to the different values of space group*

OUTLIER ANALYSIS

Our next step was to find *Outliers*(if any) in our data set. So we basically find the InterQuartile Range(IQR) for the same and the data points that were found outside the range (between mean+ 1.5IQ and Mean -1.5IQR) were outliers.



```
tr.outliers.value_counts()

False    2349
True       51
Name: outliers, dtype: int64
```

Total Outliers found : 51

SCATTER PLOTS

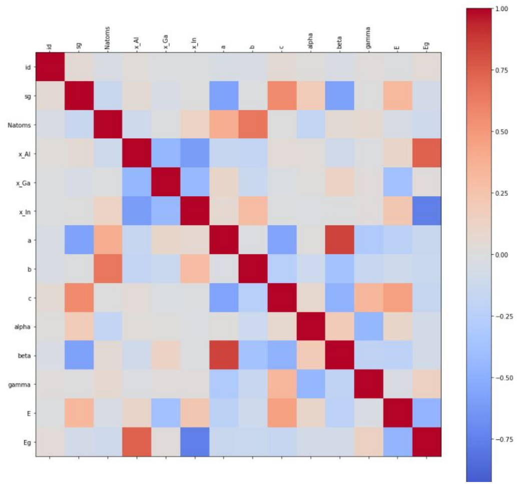
We visualize the relationship between the target properties and the different features using *scatter plots.*

Following scatter plots were analyzed.

- *relationship between our target properties and the total number of atoms.*
- *relationship between our target properties and the lattice vectors.*
- *relationship between our target properties and the lattice angles.*
- *relationship between our target properties and the percentage composition of elements.*

CORRELATION ANALYSIS

Now, our next step is to basically see whether our features interact with each other or not and if interact then how they interact. There are two aims here, namely, identifying whether any of the features strongly correlate with each other so that any one of them can be neglected from further analysis, and discovering if there is any strong correlation with the target variables.



So, now we generated a correlation matrix and saw the strength of correlation using a heat map. Then we checked for pairs of features with correlation greater than 0.9 so that we could remove one of each pair.

The only strong correlations we found was between (1) Eg and x_Al - 0.74 and (2) beta and a - 0.85.

We concluded that since there is no pair with corr. > 0.9, the corr. between any 2 features is not strong enough to remove any of them.

FEATURE ENGINEERING

As we found from scatter plots that there is no strong correlation of target variables with the existing variables. So now, we aim to engineer new features using the existing ones and hope to achieve better correlation with the target variables.

1. VOLUME OF LATTICE UNIT CELL

We calculate the value of the parallelepiped unit cell using the lengths of the lattice vectors and the lattice angles and see its relationship with the target variables.

And further analyzed: *scatter plot between our target properties and the volume.*

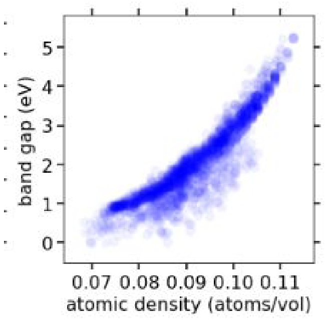
No strong correlation was found.

2. ATOMIC DENSITY

We incorporate the number of atoms in the lattice and calculate the atomic number density.

And further analyzed: *scatter plot between our target properties and the density*

A Strong correlation was found between band gap energy and the atomic density.



3. GEOMETRY.XYZ FILES

Apart from the 14 attributes to each from, there is also a geometry.xyz files corresponding to each row which contains spatial information of each material.

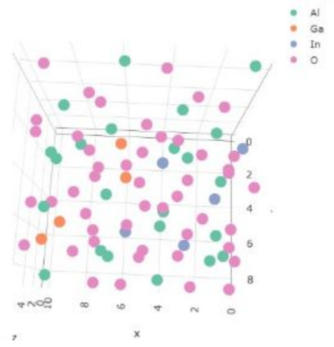
Each geometry file contains a host of information which might be difficult to utilize in further analysis. So we apply PCA to these 3-D files to convert them into linearly uncorrelated PCs which we can perhaps further use in our project.

The PCA algorithm is leveraged to project the 3D points that are contained in the xyz-files onto the 2D plane. While in the 3D space some patterns might remain unrevealed, the projection along their greatest variance might provide a more accessible view to the point configure.

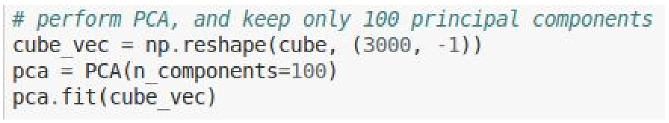
PRINCIPAL COMPONENT ANALYSIS OF GEOMETRY.XYZ FILES

We extracted atom type and position from each line to construct 3D cubes encoding the density of each atom type in the lattice cell.

We attached, around each atom, a small sphere with a Gaussian profile and a radius of the same order as the pixel size. The 3D cube value for each channel and each pixel position was then given by the sum of density of all atoms corresponding to this particular channel at this particular position.

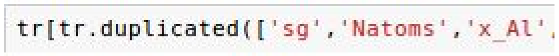


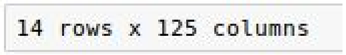
We then used PCA to compute the main 100 components over all samples and project each 3D cube onto these N components.


```
# perform PCA, and keep only 100 principal components
cube_vec = np.reshape(cube, (3000, -1))
pca = PCA(n_components=100)
pca.fit(cube_vec)
```

REMOVING DUPLICATE ROWS

After computing all the new features for each row, we now look for duplicate rows, and remove them.


```
tr[tr.duplicated(['sg','Natoms','x_Al',
```

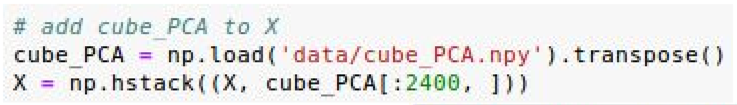

```
14 rows x 125 columns
```

We find 14 duplicate rows, which don't need to be included in further project.

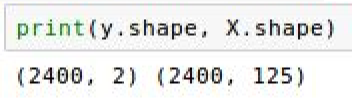
APPLYING DATA MINING ALGORITHMS

PREPARING THE DATA FRAMES

We prepare the input variables and target variables dataframes by appending .xyz PCAs to the existing .csv file.

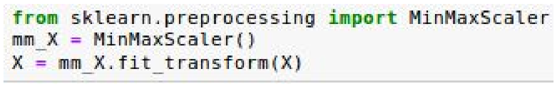

```
# add cube_PCA to X
cube_PCA = np.load('data/cube_PCA.npy').transpose()
X = np.hstack((X, cube_PCA[:2400, ]))
```

X is the input variables dataframe with 125 attributes including the 100 geometry PCs. We also perform MinMax scaling.


```
print(y.shape, X.shape)

(2400, 2) (2400, 125)
```

y is the target variables dataframe with the two target variables, formation energy and band gap energy.

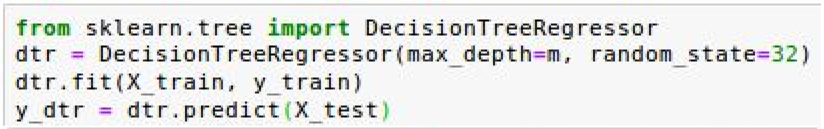

```
from sklearn.preprocessing import MinMaxScaler
mm_X = MinMaxScaler()
X = mm_X.fit_transform(X)
```

DM ALGORITHMS

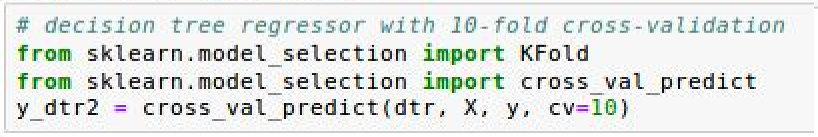
We go on applying the following DM algorithms on the above generated dataframes.

1. Decision Tree Regressor

We first run a loop to find out the optimal max_depth for our decision tree model to avoid under-fitting or over-fitting.


```
from sklearn.tree import DecisionTreeRegressor
dtr = DecisionTreeRegressor(max_depth=m, random_state=32)
dtr.fit(X_train, y_train)
y_dtr = dtr.predict(X_test)
```

We then applied the same algorithm with k-fold cross-validation instead of train_test_split.


```
# decision tree regressor with 10-fold cross-validation
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_predict
y_dtr2 = cross_val_predict(dtr, X, y, cv=10)
```

K-Fold Cross Validation

First we used train_test_split to split the data into training and validation sets, but we realised it is not really correct as some data points never come in one of test or train sets. So we moved on to k-fold cross validation, which splits the data into k sets, trains data on k-1 sets and validates using the remaining set, and does this k times.

All the algorithms from here on have been applied using k-cross validation.

2. Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor(max_depth=100, random_state=6, n_estimators=20)
y_rfr = cross_val_predict(rfr, X, y, cv=10)
```

3. Support Vector Regression

```
from sklearn.svm import SVR
svr = SVR(kernel = 'rbf', gamma=0.001)
y_svr = cross_val_predict(svr, X, y[:,0], cv=10)
```

4. Polynomial Regression

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
q=PolynomialFeatures(degree=2)
X_poly=q.fit_transform(X_train)
linreg=LinearRegression()
linreg.fit(X_poly,y_train)
y_poly = linreg.predict(q.fit_transform(X_test))
```

5. Gradient Boosting Regression

```
from sklearn import ensemble
params = {'n_estimators': 500, 'max_depth': 4, 'min_samples_split': 2,
          'learning_rate': 0.01, 'loss': 'ls'}
xgb = ensemble.GradientBoostingRegressor(**params)

xgb.fit(X_train, y_train[:,0])
y_xgb = xgb.predict(X_test)
```

6. Neural Network

We trained a neural network model. For the activation function, we tried three options, tanh, sigmoid and ReLU. ReLU gave far less R2_score than the other two. Tanh and sigmoid, though comparable, we chose tanh as our activation function due to its slightly better R2_score than sigmoid.

```
import tensorflow as tf
layers = [16] * 8
activation = tf.tanh
tf_is_training = tf.placeholder(tf.bool, None)
tf_x = tf.placeholder(tf.float32, (None, X_train1.shape[1]), name='tf_x')
tf_y = tf.placeholder(tf.float32, (None, 2), name='tf_y')
for i, n in enumerate(layers):
    l = tf.layers.dense(tf_x, n, activation=activation, kernel_regularizer=kernel_regularizer_l2
                        name='layer%s' % (i + 1))
output = tf.layers.dense(l, 2, name='output')
optimizer = tf.train.AdamOptimizer(learning_rate=0.002)
y_neural = {tf_x: X_validation1, tf_y: y_validation1, tf_is_training: False}
```

PERFORMANCE COMPARISON OF MODELS

We used the following two metrics to compare the performance of our regression models -

1. *R^2 (coefficient of determination) regression score function*

   The **r2_score** function computes the **coefficient of determination**, usually denoted as R².

   $$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

   It represents the proportion of variance (of y) that has been explained by the independent variables in the model. It provides an indication of goodness of fit and therefore a measure of how well unseen samples are likely to be predicted by the model, through the proportion of explained variance.

   Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R² score of 0.0.

2. *Mean Relative Error*

   $$\text{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i| .$$

   We first calculated mean absolute error using the **mean_absolute_error** function which is a risk metric corresponding to the expected value of the absolute error loss or L1 -norm loss.
   We then divide this value by the mean of the actual target variable values to obtain a relative quantity.

SUMMARY OF PERFORMANCE OF OUR MODELS

We calculated the above two metrics to compare and evaluate our regression models, and the results are tabulated below (for formation_energy) -

Algorithm	**R2_score**	**Mean_relative_error**
Decision Tree Regressor	0.90045845	0.14532853
Decision Tree Regressor with k-fold cross validation	0.89911417	0.03598199
Random Forest Regression	0.94429537	0.02515321
Support Vector Regression	0.51251331	0.31641051
Polynomial Regression	-984432990.2	1866.641
Gradient Boost Regression	0.95531809	0.12953969
Neural Network	0.88345416	0.372282

CONCLUSION

Based on R2 scores, the best models for formation energy and band gap energy are

(1) Random Forest Regression - for formation_energy

R2 score = 0.826700276

mean_relative_error = 0.02515321185

RMSLE = 0.001191694359

(2) Gradient Boosting Algorithm - for band_gap_energy

R2 score = 0.9553180976

Mean_relative_error = 0.070370930

RMSLE = 0.00778233864