

## **Data Science Challenge Report**

Shravankumar Hiregoudar  
shravankumarhiregoudar@gmail.com

Materials scientists are continually striving to advance their ability to understand, predict, and improve materials properties. Due to the high cost of traditional trial-and-error methods in materials research (often in the form of repeated rounds of material synthesis and characterization), material scientists have increasingly relied on simulation and modeling methods to understand and predict materials properties a priori. Materials informatics is a resulting branch of materials science that utilizes high-throughput computation to analyze large databases of materials properties to gain unique insights. More recently, data-driven methods such as machine learning have been adopted in MI to study the wealth of existing experimental and computational data in materials science, leading to a paradigm shift in the way materials science research is conducted

The idea behind this challenge is to use data to solve a canonical thermodynamics problem: given a pair of elements, predict the stable binary compounds that form on mixing given 2573 element pairs as training data. Each of the pure elemental compounds has been expanded into features using a simple application of the magpie feature set.

### **Dataset & Task:**

Training data shape: (2573,99)

Testing data shape: (750,98)

Number of features: 96

Features include properties of A and B elements like Atomic Weight, Atomic Volume, Boiling temp, Bulk modulus, row, and column in the periodic table, etc. We have 48 such properties for each element.

The task is to use these 96 features to predict the stability vector for the element pair.

## Understanding target (stability vector):

The target is the stability vector, which is the last column in training data. The format of one stability vector is '[1.0,0.0,0.0,1.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0]'. Originally, each of the target vectors is in string format. I converted all the targets into pd.DataFrame where each vector is a pd.series.

Example of stability vector: OsTi ([1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0]) translates into the following stable compounds: Os, Os{0.5}Ti{0.5} or OsTi, and Ti.

Each element is stable on its own. Every stability vector starts and ends with 1.0. For this reason, I will be predicting the stability vector without the first and last points.

## Data preprocessing:

### 1. Data Enhancing version 1 – Duplicate data

The given data contains a pair of elements and their properties. When performing ML study, we must be mindful of our data set size. We must ensure that our data set size is large enough and includes most examples of the combinations of material compositions in the material space you want to study. It is also important to consider data balance or bias in data sets. Does your data form clusters based on chemical formula, test condition, structure type, or other criteria? Are some clusters considerably over or under-represented? Many statistical models used in ML are frequentist in nature and will be influenced by data set imbalance or bias. Visualization techniques may be useful in investigating data set imbalance and bias.

The data is enhanced using the same original data. By merely reversing the order of the stability vectors for each pair and by replacing A-column information with B-column information, the data was doubled. Since there were only 2572 data points, this seemed like a worthwhile thing to do. I think this transformation perhaps also helped to capture some of the sequencings between elements in output stability vectors that I have already mentioned. Additionally, it would allow me to have

larger test, validation, and training sets to work with. It was also clear that this mirroring technique improved some of the class imbalances that existed.

Original data:

Formula A	Formula B	A_atomicvol	B_atomicvol	A_atomicwt	B_atomicwt
Ac	Ag	37.43309	17.07565	227	107.8682

The duplicate data added to original data to enhance:

Formula A	Formula B	A_atomicvol	B_atomicvol	A_atomicwt	B_atomicwt
Ag	Ac	17.07565	37.43309	107.8682	227

The data is enhanced, we have 5144 records (5144 element pairs as training data). This version of the data is in CITRINE\enhancedData\Train\trainEnhancedV1-AddedDuplicates.csv

## 2. Data Enhancing version 2 – Add features

Feature engineering is important for smaller data set sizes and can contribute to a large model performance increase if the features are well-engineered. A common way to add more features for ML studies is by performing mean and difference on the features of A and B. As there is no direct rule in predicting the stability vector, the more features we have, the better the predictions might be.

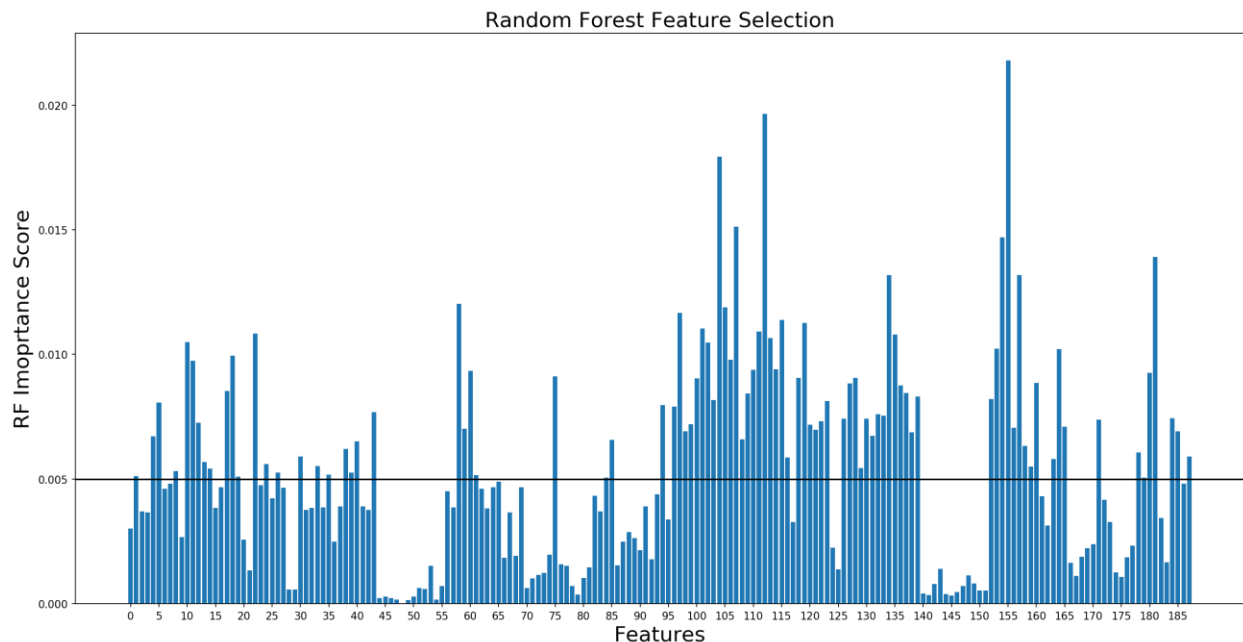
These hand-crafted features are used to predict the stability vector. The new features can be 'avg\_elements\_AtomicVolume', 'diff\_elements\_AtomicVolume', 'avg\_elements\_AtomicWeight', 'diff\_elements\_AtomicWeight', 'avg\_elements\_BoilingT', 'diff\_elements\_BoilingT', 'avg\_elements\_BulkModulus', 'diff\_elements\_BulkModulus', 'avg\_elements\_Column', 'diff\_elements\_Column'

After a quick study I found, the atomic radii, covalent radii, miracle radii play an essential role in forming bonds. So, its important consider the average of these features and difference of property of A and B. Now we have total of 187 features

We will see how these additional features gave us the better results in the next part.

## Random Forest Feature Selection:

Random forest consists of several decision trees. Every node in the decision trees is a condition on a single feature, designed to split the dataset into two so that similar response values end up in the same set. The measure based on which the (locally) optimal condition is chosen is called impurity. For classification, it is typically either Gini impurity or information gain/entropy and for regression trees it is variance. Thus, when training a tree, it can be computed how much each feature decreases the weighted impurity in a tree. For forest, the impurity decrease from each feature can be averaged and the features are ranked according to this measure. Feature importance measure exposed in sklearn's Random Forest implementation. In our study, all the features are fed in and we get the importance of each features.



*Fig.1. RF Feature importance with default RF parameters*

In the above graph, the x axis is feature number and y axis is the importance score associated with that feature. The higher the importance score, the better the feature. The line at 0.005 is the threshold for considering the feature as important. In case of, the feature that was not discarded with its corresponding pair was formulaA\_elements\_NdUnfilled, although its counterpart for element B was not significantly above threshold. Therefore, I also discarded formulaB\_elements\_NdUnfilled.

The feature numbers till 96 are the original features given in the dataset. The feature number from 97-187 is the hand-crafted ones. The most important 10 features were,

- a. diff\_elements\_MendeleevNumber
- b. avg\_elements\_Electronegativity
- c. avg\_elements\_Column
- d. diff\_elements\_CovalentRadius
- e. avg\_elements\_MendeleevNumber
- f. diff\_elements\_Polarizability
- g. diff\_elements\_MiracleRadius
- h. avg\_elements\_HeatCapacityMolar
- i. formulaA\_elements\_MendeleevNumber
- j. diff\_elements\_Column

9 out of top 10 features were the derived features. This was a great news. The difference in the Mendeleev number of A and B was the most important feature. The derived features were having high importance score than the original feature. For example, The Avg of electronegativity of A and B is the second important feature but the individual electronegativities were having very low important score.

Using a threshold of 0.005, I removed 116 features out of 188. Now, we have the 72 features which has importance score greater than 0.005.

## Data Normalization:

It is beneficial to scale your input data (X). For a regression task, it may also be helpful to scale the targets (y) as well. Scaling can be done in many ways. Often, the input data is scaled to have zero-mean and unit variance. This allows for more stable gradients and faster model convergence since the resulting feature dimensions are similar in scale.

This is done by using the transformation:

$$X' = X - \bar{X} / \sigma_X$$

where  $\bar{X}$  denotes the mean and  $\sigma_X$  the standard deviation of X. In some cases, applying the logarithm function to your values before scaling them according to eq 1 may further improve your model performance.

## Train/Test/Validation split:

Further, I split the data into three data sets: train(60%), validation(10%), and test(30%). The split should be performed in a reproducible way (e.g., by assigning a random seed and shuffling the data set); Make sure that no same (or similar) data appear in the test data set, if they are already present in the train or validation data set.

For example, if you have several properties of a chemical compound that are performed at different measurement conditions in the train data set (e.g., temperature or pressure), during the testing phase, your model would likely perform well if it is asked to predict the property of the same compound at a different condition. This, however, gives you an inflated estimate of how well the model will generalize in cases where it has not seen a chemical compound before. For a truly rigorous evaluation of your model's generalization performance, you should take care to avoid this data leakage when you split your data.

During the training stage, models are shown the training data as part of the learning process. Validation data is used to assess and tune different model hyperparameters and to compare with the predictions of different model/hyperparameter combinations to evaluate a model's performance. In

contrast, test data is used to evaluate a model's performance as a final step, after the model has been finalized. Models must not be trained nor tuned on the test data set. Use the same train, validation, and test data sets for all modeling and model comparison/ benchmarking steps.

## Choosing Model:

The data set size will almost always determine your available choices of ML models. For smaller data set sizes, classical and statistical ML approaches (e.g., regression, support vector machines, k-nearest neighbors, and decision trees) are more suitable. In contrast, neural networks require larger amounts of data and only start becoming feasible/useful when you have training data points on the order of thousands or more. Typically, ML models such as regression, decision tree/ random forest, k-nearest neighbors, and support vector machines are used on smaller data sets. These algorithms can be further improved by applying bagging, boosting, or stacking approaches. There are many existing Python libraries for implementing the above, with perhaps the most well-known being scikit-learn. For larger data sets, neural networks and deep learning methods are more commonly used.

In our case we compare Random Forest Classifier, Support Vector Classifier and Logistic Regression algorithms. Trained models are compared by evaluating their performance on the held-out validation data set using computed test metrics such as accuracy, precision, recall, F1- score

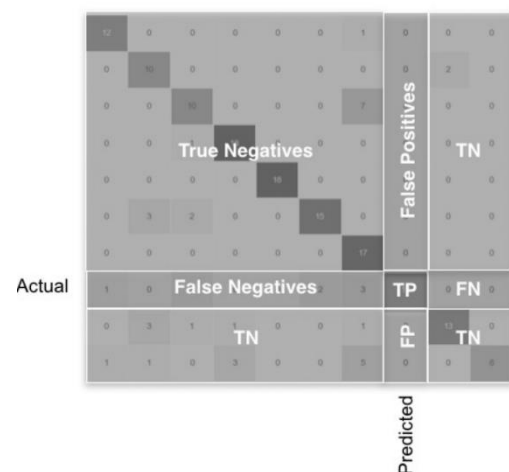
*Understanding the terms:*

*True Positive (TP): It refers to the number of predictions where the classifier correctly predicts the positive class as positive.*

*True Negative (TN): It refers to the number of predictions where the classifier correctly predicts the negative class as negative.*

*False Positive (FP): It refers to the number of predictions where the classifier incorrectly predicts the negative class as positive.*

*False Negative (FN): It refers to the number of predictions where the classifier incorrectly predicts the positive class as negative.*



A confusion matrix diagram illustrating classification results. The matrix is a 2x2 grid with 'Actual' on the left and 'Predicted' on the right. The top row represents the 'Actual' positive class, and the bottom row represents the 'Actual' negative class. The left column represents the 'Predicted' positive class, and the right column represents the 'Predicted' negative class. The cells are labeled as follows: True Positives (TP) in the top-left, True Negatives (TN) in the top-right, False Positives (FP) in the bottom-left, and False Negatives (FN) in the bottom-right. The matrix is shaded with a light gray background and a darker gray border around the cells.

	Actual Positive	Actual Negative
Predicted Positive	TP	FP
Predicted Negative	FN	TN

*Accuracy: It gives you the overall accuracy of the model, meaning the fraction of the total samples that were correctly classified by the classifier. To calculate accuracy, use the following formula:  $(TP+TN)/(TP+TN+FP+FN)$ .*

*Precision: It tells you what fraction of predictions as a positive class were positive. To calculate precision, use the following formula:  $TP/(TP+FP)$ .*

*Recall: It tells you what fraction of all positive samples were correctly predicted as positive by the classifier. It is also known as True Positive Rate (TPR), Sensitivity, Probability of Detection. To calculate Recall, use the following formula:  $TP/(TP+FN)$ .*

*F1-score: It combines precision and recall into a single measure. Mathematically it is the harmonic mean of precision and recall.*

The results of these algorithms with default parameters were as follows,

Model	Accuracy			Validation set results (weighted)			Micro
	Train	Validation	Test	Precision	Recall	F1 score	F1 score
Logistic Regression	0.53858	0.535622	0.522222	0.42	0.56	0.45	0.56
RF classifier	0.994753	0.610751	0.586111	0.52	0.58	0.54	0.58
Support Vector Classifier	0.576543	0.587435	0.577778	0.43	0.59	0.49	0.59

*Table.1. Algorithm comparison with default parameters*

*Note that for “micro”-averaging in a multiclass setting with all labels included will produce equal precision, recall and F, while “weighted” averaging may produce an F-score that is not between precision and recall.*

Hurray! My second favorite algorithm, Random Forest outperforms My favorite, SVC. Random forest is one of the most well-known ensemble methods for good reason – it is a substantial improvement on simple decision trees. Now, we decided the algorithm. Next step is to decide the parameters for random forest.



## Hyper parameter tuning for RF:

Random Forest Hyperparameters we will be Looking at:

### max\_features:

This resembles the number of maximum features provided to each tree in a random forest. We know that random forest chooses some random samples from the features to find the best split. Let us see how varying this parameter can affect our random forest model's performance.

max_features	n_estimators	Precision	recall	F1
log2	10	0.676768	0.39645	0.5
log2	20	0.72	0.426036	0.535316
log2	30	0.735577	0.452663	0.56044
log2	50	0.747475	0.43787	0.552239
log2	100	0.734597	0.45858	0.564663
log2	500	0.736842	0.455621	0.563071
sqrt	10	0.665	0.393491	0.494424
sqrt	20	0.703518	0.414201	0.521415
sqrt	30	0.726829	0.440828	0.548803
sqrt	50	0.741627	0.45858	0.566728
sqrt	100	0.743842	0.446746	0.558226
sqrt	500	0.738318	0.467456	0.572464

*Table.2. Understanding max\_features results*

The possible value of max\_features can be auto, sqrt or log2.

- If "auto", then max\_features=sqrt(n\_features).
- If "sqrt", then max\_features=sqrt(n\_features) (same as "auto").
- If "log2", then max\_features=log2(n\_features)

Here, the average F1 score for log2 max\_features is 0.54595 and for sqrt is 0.543676. Since, avg of F1 is higher for log2. We will choose **log2** as our best max\_features

Min sample leaf:

This Random Forest hyperparameter specifies the minimum number of samples that should be present in the leaf node after splitting a node. 1 is the default number for this parameter.

min_samples_leaf	n_estimators	Precision	recall	F1	Avg F1
16	10	0.692913	0.260355	0.378495	0.379575
16	20	0.741379	0.254438	0.378855	
16	30	0.761062	0.254438	0.381375	
8	10	0.664671	0.328402	0.439604	0.439286667
8	20	0.722222	0.307692	0.431535	
8	30	0.726667	0.322485	0.446721	
4	10	0.693548	0.381657	0.492366	0.495806333
4	20	0.702247	0.369822	0.484496	
4	30	0.726776	0.393491	0.510557	
2	10	0.691099	0.390533	0.499055	0.520988333
2	20	0.726804	0.41716	0.530075	
2	30	0.731959	0.420118	0.533835	
1	10	0.698113	0.43787	0.538182	0.536448
1	20	0.727723	0.434911	0.544444	
1	30	0.741935	0.408284	0.526718	

Table.3. Understanding min\_samples\_leaf results

RF performs best when min\_samples\_leaf is 1. As It gives the highest Avg F1 score.

Min sample split:

This parameter that tells the decision tree in a random forest the minimum required number of observations in any given node to split it. The default value of the minimum\_sample\_split is assigned to 2. This means that if any terminal node

has more than two observations and is not a pure node, we can split it further into subnodes.

Having a default value as 2 poses the issue that a tree often keeps on splitting until the nodes are completely pure. As a result, the tree grows and therefore overfits the data.

min_samples_split	n_estimators	Precision	recall	F1	Avg F1
32	10	0.731343	0.289941	0.415254	0.40763867
32	20	0.706767	0.278107	0.399151	
32	30	0.727273	0.284024	0.408511	
16	10	0.719745	0.33432	0.456566	0.45718167
16	20	0.717949	0.331361	0.453441	
16	30	0.730769	0.337278	0.461538	
8	10	0.668508	0.357988	0.466281	0.48496933
8	20	0.679144	0.37574	0.48381	
8	30	0.723757	0.387574	0.504817	
4	10	0.62963	0.402367	0.490975	0.51496633
4	20	0.735751	0.420118	0.53484	
4	30	0.731183	0.402367	0.519084	
2	10	0.714286	0.428994	0.536044	0.54252767
2	20	0.705314	0.431953	0.53578	
2	30	0.727273	0.449704	0.555759	

Table.4. Understanding min\_samples\_split results

RF performs best when min\_samples\_split is 2. As It gives the highest Avg F1 score.

#### n\_estimator:

We know that a Random Forest algorithm is nothing but a grouping of trees. But how many trees should we consider? We might say that more trees should be able to produce a more generalized result, right? But by choosing a greater number of trees, the time complexity of the Random Forest model also increases. This means that choosing many estimators in a random forest model is not the best idea. Although it will not degrade the model, it can save

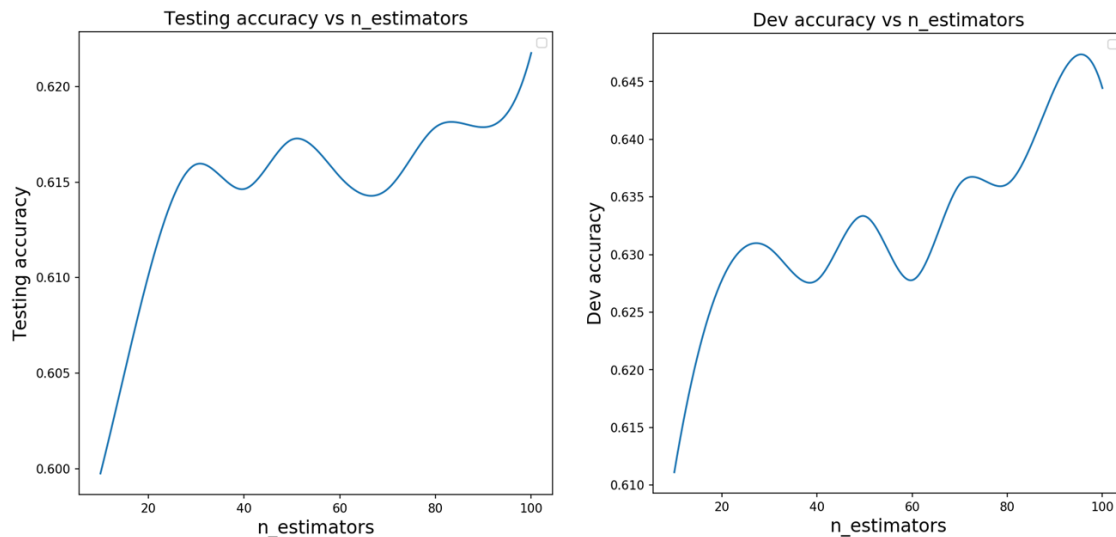
you the computational complexity and prevent the use of a fire extinguisher on your CPU!

n_estimators	Precision	recall	F1	Train Accuracy	Test Accuracy	Dev Accuracy	Time
10	0.706806	0.399408	0.510397	0.948148	0.599741	0.611111	1.527694
20	0.743455	0.420118	0.536862	0.983642	0.610104	0.627778	2.103511
30	0.737624	0.440828	0.551852	0.993827	0.615933	0.630556	2.998105
40	0.712121	0.41716	0.526119	0.997531	0.614637	0.627778	4.003864
50	0.745098	0.449704	0.560886	0.998148	0.617228	0.633333	5.61473
60	0.733668	0.431953	0.543762	0.999074	0.615285	0.627778	7.241335
70	0.719626	0.455621	0.557971	0.999691	0.614637	0.636111	7.6074
80	0.722488	0.446746	0.552102	0.999691	0.617876	0.636111	8.962361
90	0.746411	0.461538	0.570384	1	0.617876	0.644444	9.480747
100	0.736111	0.470414	0.574007	0.999691	0.621762	0.644444	11.208945
200	0.730769	0.449704	0.556777	1	0.621114	0.636111	18.596219
500	0.725118	0.452663	0.557377	1	0.620466	0.638889	47.842952
1000	0.733333	0.455621	0.562044	1	0.618523	0.641667	116.872005

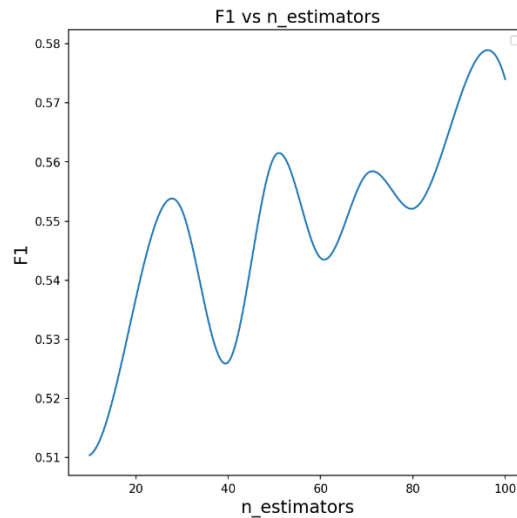
Table.5. Understanding n\_estimator results

max_features	max_depth	criterion	min_samples_leaf	min_samples_split	bootstrap
log2	None	gini	1	2	TRUE

Table.6. Finalized parameters



*Fig.2. Variation of accuracy and test and dev accuracy with  $n\_estimators$*



*Fig.3. Variation of F1score with  $n\_estimators$*

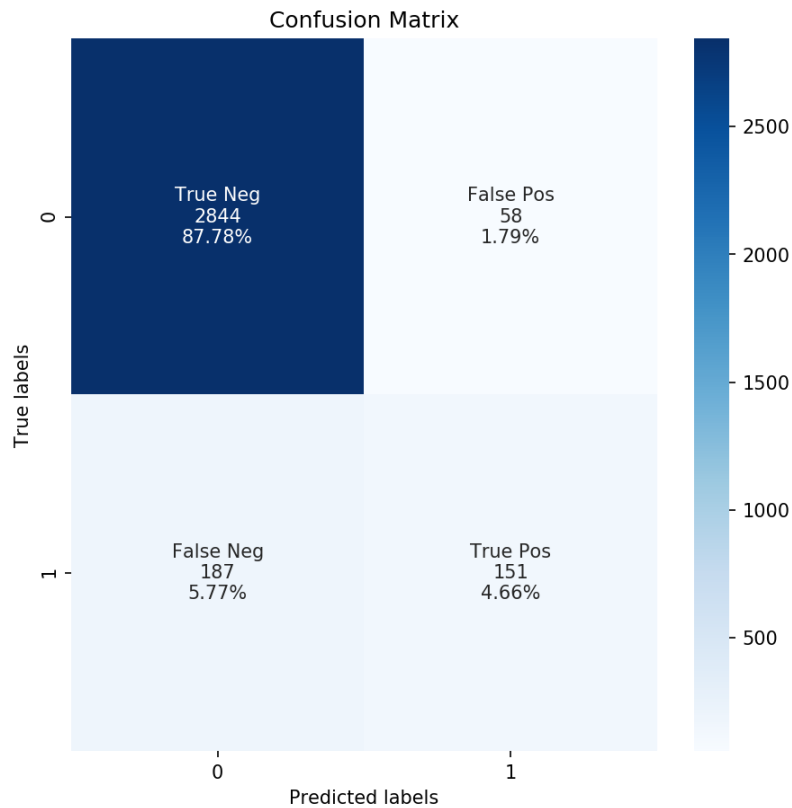
Now, the final hyperparameter is the  $n\_estimator$ . The F1 increases as the  $n\_estimator$ . But higher  $n\_estimator$  corresponds to higher runtime and complexity. We must find a balance between all these things. Our algorithm should give high accuracy and F1 in low CPU time. *I have a 5years old laptop, so the time taken is slightly higher than usual.*

By choosing  $n\_estimators$  as 50, We achieve training accuracy of 99.81%, Testing accuracy of 61.7% and Validation set accuracy of 63.33%. The model gives us **0.56** F1 score in considerably low time.

## Results:

N_estimator	max_features	max_depth	criterion	min_samples_leaf	min_samples_split	bootstrap
50	log2	None	gini	1	2	TRUE

With above parameters our mode gives the best results.



*Fig.4. Confusion matrix*

True Negatives (Right predictions) - 2844

False Positives (Type I Error) - 58

False Negatives (Type II Error) - 187

True Positives (Right predictions)- 151

This model is used for prediction on test dataset. The result of predictions is written in the last column of the dataset and store in CITRINE\results\predictions\_enhnacedTestData.csv

In case if you just need the original test data results. Consider the first 750 rows of predictions\_enhnacedTestData.csv file.

## Conclusion:

While various machine learning methods, including classical methods and more advanced techniques such as deep learning and neural network-based architectures, have successfully been used for the prediction of materials properties, unique challenges still exist for their application in the domain of materials informatics. There are common pitfalls in the gathering, analysis, and reporting of materials science-related data and machine learning results and in the facilitation of reproduction studies.

This was a fun learning experience working with material science data. I completely enjoyed the whole process of this data challenge. My model should give a good metrics on the test dataset. Since, the parameters are tuned well with enough data size.

## Files attached:

- Results/predictions\_enhancedTestData.csv  
*The final test set results. Last column contains the predictions*
- Results/ RFHyperparameterTuning.xlsx  
*RF Hyperparameter results. Variation of F1 score with change in parameters*
- Problem statement/ and originalData /  
The original problem statement and train, test data provided by citrine
- Modules/  
*Containing 3 python functions*
- enhancedData/  
*Contains the enhanced versions of data. Refer to data preprocessing above.*  
*V1 = Added Duplicates*  
*V2 = Added Duplicates + Added features*  
*Final = Added Duplicates + Added features + Important features only*