Binary alloys challenge

Before starting my summary I would like to make a few general remarks:

- To address this challenge I tried a huge number of different approaches and I also rediscovered ideas that were actually already in the literature. Below I summarize the main points in chronological order. I provided also a few Jupyter notebooks corresponding to part of this work (their names are highlighted in red in this document).
- The challenge focuses on the prediction of the full stability vector. This is a difficult problem as the quality of the prediction is evaluated by subset accuracy (https://en.wikipedia.org/wiki/Multi-label_classification#Statistics_and_evaluation_metrics). Using this metric it is enough to make a single mistake in one of the 9 labels (the length of the stability vector minus the two pure elements) to have a "wrong" prediction for that compound. Accordingly, it is hard to reach a high score. Still, a model can be quite predictive in terms of single compound stability. To show this I will print also other metrics in my notebooks (precision, recall, f1). In my final prediction I expect an accuracy of 60-62%, which is about 8-10% above simply considering all the compounds as unstable.

The challenge concerns the prediction of the stability of binary compounds. Two files are provided:

- training_data.csv: The training dataset, that contains several features, including the name of the elements composing the compound and several other features describing their properties. The name of the features are rather self-explanatory: formulaA_elements_AtomicVolume, formulaA_elements_AtomicWeight, formulaA_elements_BoilingT, formulaA_elements_BulkModulus, etc. For a few others, such as formulaA_elements_ElectronSurfaceDensityWS or formulaA_elements_HHIp, I had to do some searching. The labels associated with the features correspond to a discretization of the 1D binary phase diagram at 10% intervals. For example, the label for 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.
- test_data.csv: The compounds whose stability needs to be predicted. The features are the same as before but of course no labels are provided.

Before describing the different attempts to address the problem, I will discuss more in detail the structure of the problem. The labels to be predicted are given in the form [1.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,0.0,0.0,1.0,0.0,0.0,1.0] corresponding to pairs of elements formulaA and formulaB. Specifically, the labels correspond to the 1D binary phase diagram: [100% element A, (90% element A-10% element B), (80% A-20% B), (70% A-30% B), (60% A-40% B), (50% A-50% B), (40% A-60% B), (30% A-70% B), (20% A-80% B), (10% A-90% B), 100% B] where a 1 indicates stability and a 0 indicates instability of the corresponding compound. This data format translates naturally into a multilabel problem. However, at a later time I also addressed

the challenge as a binary classification problem. As all the elements are considered as stable in their pure form in general I will drop 100% A and 100% B (first and last label).

This is what I did (roughly in chronological order):

1-Data analysis: I started by a general analysis of the data and I verified data integrity. For example, I showed that noble gases do not form any compounds; this is indeed expected as these elements are chemically inert. I also noticed that while 70%-30% (or vice versa) compounds are the most frequent, the 90%-10% mixture is rare. Accordingly, the number of training examples per mixture percentage (stoichiometry) is different and so also the "accuracy" for the different classes is expected to be different. I also showed that some of the most recurrent elements in the test set (the set whose labels will be predicted) are among the least recurrent in the training set; again this could be a problem as I have to make more predictions for elements that have the smallest number of training examples. The details can be found into the notebook data_analysis.ipynb.

2 - Multilabel classifier and symmetry in the data: With a "quick and dirty" approach I started to see what I could do with the data provided. I approached the problem as a multilabel (9 labels) problem using random forest and up to 2 layers fully connected neural network (relu activation for the hidden layers and sigmoid for the output layer with 9 nodes for 9 labels). Alternatively, I considered also logistic regression and support vector classifier with a one vs rest approach; the results in this case were very disappointing because this type of approach trains a different classifier for each class and does not keep into account the correlation between labels. In this phase of the project my purpose was only to see the maximum level of accuracy (subset accuracy, namely accuracy on all the 9 labels) that could be reached: I kept aside a hold out test set of 10% of the data and tuned the model parameters to improve the subset accuracy of the model on the hold out test set. This is of course a bad practice as I'm overfitting to the the hold out test set. However, at this stage I was just exploring and not building a real predictive model. The algorithm that could best perform was the random forest with about 65-66% of subset accuracy on the hold out test set. Again, this is an optimistic estimate since I'm overfitting the hold out data and still the accuracy is relatively low. To obtain these results I used the basic features provided with the challenge and at most I introduced one-hot-encoding features for the elements names (this did not help the model). For the sake of completeness I will mention that the neural network did not pass 61%-62% subset accuracy.

A problem in particular attracted my attention. Each list depends on the order of the elements. For example, if the pair Ac-Tl corresponds to [1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0] then Tl-Ac would correspond to [1.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0], which is the inverse of the previous diagram. For this reason I decided to add to the 2572 training examples other 2572 obtained swapping the order of the elements and of the corresponding features and inverting the

diagram. I realized soon that none of my models understood the basic "grammar" of the data. For example, if the model was trained reaching a high level of accuracy on the pairs A1-B1, A2-B2, A3-B3, etc., it was then failing on the pairs B1-A1, B2-A2, B3-A3, etc. in the hold out test set. I would like to mention that: (1) The ordered pairs I just mentioned B1-A1, B2-A2, B3-A3, etc. were part of the much smaller hold out test set; accordingly in the training set there were several examples of swapped pairs to train the model on; (2) Of course I tuned the parameters, this was not simply a problem of overfitting.
To give a stronger "directionality" to the features I tried to introduce new features as the ratio featureA/featureB for A-B and featureB/featureA for B-A or as the signed difference featureA-featureB for A-B and vice versa for B-A. Then I also tried to go to higher order features (quadratic and cubic) but nothing worked. The best subset accuracy I obtained was 67% (overfitting the hold out test set).

To build a model that could describe the "grammar" of the problem I also tried a seq2seq model, which is based on recurrent neural networks. I followed this tutorial to build an encoder decoder model that could be trained to translate from one language to another. My purpose was to translate A-B to a string of zeros and ones keeping into account the AB/BA grammar. I tried to represent an element A by using either the full sequence of features or by using a one-hot-encoding of the element names (this second approach is similar to what is done for language translation). This deep learning model was either overfitting or underfitting the data and for sure did not learn the swapping grammar. In general, deep learning models require a large number of training examples.

Final note: These are my thoughts about the element swapping at the end of the project.
I think that this issue is largely related to (1) the limited number of training examples; (2) the fact that this challenge is related to subset accuracy, namely the accuracy on the full stability vector prediction; this is a very strict metric and reaching high values is difficult. To make the element swapping problem more mild is important to train the model both with an element pair and the corresponding swapped pair with the inverted label.

3 - A simplified problem: Up to now I had focused on the problem of symmetry in the data, which is important but only to a certain extent related to the accuracy of the model. Namely, having a model that respects the A-B/B-A grammar does not automatically imply that the model is also accurate. For this reason I decided to consider a simplified model. Specifically, I symmetrized the labels to create a list (or array) with only 5 entries [(90% A-10% B) or (10% A-90% B), (80% A-20% B) or (20% A-80% B), (70% A-30% B) or (30% A-70% B), (60% A-40% B) or (40% A-60% B), (50% A-50% B)]. For example, within this formulation I transformed [1.0,0.0,0.0,1.0,0.0,1.0,1.0,0.0,1.0,0.0,1.0] into [1.0,1.0,1.0,1.0,0.0]. These labels cannot be used for the final prediction as they contain less information. However, they are not affected by the problem of the element order as formulaA-formulaB and formulaB-formulaA

correspond exactly to the same labels. This simplified problem is meant to let us understand the (maximum) level of subset accuracy that we can hope for by using the features provided by the challenge. I used only the random forest classifier which is a quite powerful algorithm and can be trained for multilabel problems. I put aside a 10% hold out test set to verify my model at a later time. I used 10-fold cross validation and subset accuracy as scorer to optimize the parameters in the random forest model. In crossvalidation I reached a subset accuracy of 0.642+/-0.057 and a subset accuracy of 0.67 on the hold out test set. On the hold out test set I had precision of 0.78 and recall of 0.61.

This simplified model inspired me to introduce new features as weighted averages of the properties of the elements. Specifically, I introduced features of the type 0.9*featureA+0.1*featureB, 0.1*featureA+0.9*featureB, 0.8*featureA+0.2*featureB, etc. I considered several choices for the specific featureA and featureB. I noticed that the derived average features were helping to improve the accuracy of certain labels and deteriorating the accuracy of others. As a matter of fact in a multilabel problem the algorithm might not understand that a certain feature is meant to be used only to improve the prediction of a specific label. For example, the derived feature 0.9*featureA+0.1*featureB can influence not only the mixture (stoichiometry) 90%A-10%B but also the mixture 50%A-50%B. To solve this problem I thought about training a different model with different features for each different label (namely one of the 5 labels). I understood rapidly that this was not a suitable idea as I would have lost the correlation between labels and I would have risked to create a series of models completely disconnected. In order to exploit compound (elements+stoichiometry) specific features, such as the weighted averages, I developed the idea presented in the next section. Soon after I realized that this approach was the "standard" in the scientific literature in the field.

Details and code can be found in simplified_problem.ipynb

4 - Binary classifier and compound specific feature engineering: So far I adressed the challenge as a multilabel classification problem. Here I will show how it can be reformulated as a binary classifier. Specifically, we can unroll the vector of labels (for example [0.0,0.0,1.0,0.0,1.0,1.0,0.0,1.0,0.0]) into a column; now each row, namely each training example, has a single label that can take values 0 or 1. In this way I will get a set of training data that is much larger than the original one, with 25729 *(n of original training data * n of labels*) training examples. Also, within this reformulation the problem can be addressed as a binary classification. However, there is a problem: How can I create compound (stoichiometry) specific features? For example, now the (70% A-30% B) compound becomes an "independent" training example and ideally should have at least some specific descriptors to distinguish it from the (60% A-40% B) compound in the same stability vector. Using my intuition I designed features such as differences |featureA-featureB| and weighted averages 0.1*featureA+0.9*featureB for the

(10% A-90% B) compound. Then I discovered that there was a rich literature in this field. This will be discussed in details below.
The reformulation of the problem as a binary classifier has a few potential advantages:

- We have more training examples;
- Differently from multilabel classification, there is plenty of algorithms that can be used as binary classifier
- Introducing compound specific features might boost the "accuracy"
- The problem of the swapping of the elements I previously analized is less crucial in this model
- In a way this reformulation feels more natural

In the feature engineering part I considered the following features:

- The original set
- Dummy variables corresponding to the different mixture classes (stoichiometries) 90%-10, 80%-20%, etc.
- The actual mixing percentage: For example for a 90%-10% mixture we will have a column percentage_A with 0.9 and a column percentage_B with 0.1
- Features correponding to the L^p norms of the stoichiometry. These are the stoichiometric attributes in supplementary material of Ward et al. npj Computational Materials 2, 16028 (2016).
- I (re)discovered the differences and weighted average features on my own. However, they were already used together with average deviations in Ward et al. npj Computational Materials 2, 16028 (2016).
- Features corresponding to the valence orbital occupation attributes. These features have been proposed in Meredig et al. PRB 89 094104 (2014).

Following Ubaru et al. PRB 95, 214102 (2017) I also introduce some new element features which were missing in the data provided: Cohesive energy, Enthalpy of Vaporization and Electron Affinity.
At this point I had 281 features. To avoid overfitting and spare computer time I included only 100 features selected by the random forest algorithm. With the exception of the Mendeleev Number all the other features I kept were engineered features.

I trained different machine learning algorithms as binary classifier for the stability problem. I put aside a 10% hold out test set to verify my model at a later time. By using 10-fold cross validation on the training set I performed a grid search for the best parameters for all these classifiers. I used as scoring function the F1 function. Optimizing this metric leads to a good compromise between precision and recall. This is not exactly like training on subset accuracy (accuracy on

the full stability vector) but it's a reasonable choice. With the optimized parameters I then evaluate the performance on the hold out (test) set. These were my results:

- Random forest classifier: In 10-fold cross validation I obtained 0.553 (+/-0.073) as best f1 score. On the hold out test set I obtain a 0.63 f1 score (on the stable compounds) with a precision that is sizeably higher than the recall. The subset accuracy on the hold out test set is 0.64.
- Support vector classifier: In 10-fold cross validation I obtained 0.622 (+/-0.045) as best f1 score. On the hold out test set I obtain a 0.69 f1 score (on the stable compounds) with a precision that is comparable to recall. The subset accuracy on the hold out test set is 0.635, slightly worst than RF.
- Neural network: Parameter optimization was computationally expensive and for this reason I used only a single hidden layer. In 10-fold cross validation I obtained 0.606 (+/-0.032) as best f1 score.
- Others: I also rapidly tried the k-nearest neighbours and Gaussian naive Bayes classifiers without a sizeable improvement in the results (not shown in the notebook mentioned below).

Overall the performance of all the three algorithms is similar. I guess that this is due to a good work in the feature engineering that "leaves less work" to the algorithms. The classifier that seems to perform better is the SVC.

However, I would like to highlight a few concerns about these models:

- The models are trained using f1 as metric. While this is a reasonable choice the results in the subset accuracy can have sizeable fluctuations.
- Training these models on subset accuracy (the full stability vector) is both technically difficult and conceptually not well defined. Indeed, now each compound with its own stoichiometry becomes to a certain extent an independent system.
- In a way this approach might loose the dependence between the labels in the same stability vector. For example, the stability of the (90% A-10% B) compound might have implication on the stability of (80% A-20% B) and this is only partially kept into account.

In conclusion, I believe that the models in this notebook are more suitable if the target is to find single stable compounds rather than predicting full stability vectors.

Details are provided in the binary_classifier.ipynb notebook.

5 - Final model: Based on the previous work I believe that a multilabel classifier is a better approach to predict a full stability vector. Multilabel random forests (or neural networks) “naturally” keep into account the correlation between labels. For example, the stability of the (90% A-10% B) compound might have implication on the stability of (80% A-20% B). From a practical point of view it is easy to train this type of model to optimize subset accuracy, namely accuracy in the full stability vector.

In this model I will use the compound specific features already used in the binary_classifier.ipynb notebook. However, in this case I will not "unroll" the data. For example, I will introduce the weighted average features 0.9*featureA+0.1*featureB for the (90%A-10%B) compound and similar ones for other stoichiometries. Differently from the previous work on the binary classifier, a feature such as 0.9*featureA+0.1*featureB will contribute to the full stability vector prediction and not only the corresponding stoichiometry.

For this model I used more than 900 features; they are all engineered and do not depend on the element order. This large number of features is suitable for a random forest classifier but not ideal for a neural network. For this reason I trained only a random forest classifier.

I kept a 10% hold out test set for verification. By using 10-fold cross validation on the training set I performed a grid search for the best parameters for the random forest classifier. I used subset accuracy as scoring function. In 10-fold cross validation I obtained 0.623 (+/-0.064) as best subset accuracy. On the hold out test set I obtain a 0.62 f1 score with a precision (0.80) that is sizeably higher than the recall (0.53). The subset accuracy on the hold out test set is 0.65. As a reference, by considering all the labels as equal to 0 the subset accuracy would be ~52%. Subset accuracy is a very strict metric as it indicates the percentage of samples that have all their labels classified correctly. As shown in the notebook certain classes have sizeably lower f1-score and this has implications for the overall subset accuracy.

I finally used this model to make a prediction for the data in test_data.csv. **The final results are in final.csv.**

Details are provided in the final_model.ipynb notebook.

Some references on the topic (in sparse order)**:**

1. B. Meredig et al., Phys. Rev. B **89** 094104 (2014)
2. L. Ward et al., npj Computational Materials **2**, 16028 (2016)
3. S. Ubaru et al., Phys. Rev. B **95** 214102 (2017)
4. P. Villars et al., Journal of Alloys and Compounds **317-318** 26-38 (2001)
5. C. Y. Yu et al., Scientific Reports 3:2124 (2013)
6. V. Stevanovic et al., Phys. Rev. B **85**, 115104 (2012)
7. A. O. Oliynyk et al., Chemistry of Materials **28**, 7324 (2016)
8. G. Hautier et al., Chemistry of Materials **22**, 3762 (2010)
9. C. C. Fischer, PhD Thesis