CS5014 P1

160004864

11 March 2020

1 Introduction

The aim of this practical was to use data taken from an existing dataset and create a regression model to predict output on a set of inputs and evaluate its performance. Specifically, the dataset is documented in a paper that was provided to us[1] and it describes features extracted from the chemical formula of various superconductors. Based on these features, we were required to build a linear regression model to predict the superconductor's critical temperature[2].

2 Loading and cleaning the data

2.1 The Dataset

The dataset was provided to us in a comma-separated values (csv) file, `train.csv`. It contains 80 features extracted from 8 variables based on the superconductors' elemental properties, namely Atomic Mass, First Ionization Energy, Atomic Radius, Density, Electron Affinity, Fusion Heat, Thermal Conductivity and Valence. Furthermore, an additional column describes the number of elements present in a superconductor and the last column holds the superconductor's critical temperature.

2.2 Loading the data

The dataset is loaded into a Python program using the `read_csv` method of the `pandas` module[3]. This method reads a csv file into a `DataFrame`, which is a data structure which is able to hold two-dimensional, size-mutable and potentially heterogeneous tabular data[4]. It is the primary `pandas` data structure and it provides a lot of functionality for analysing, manipulating and transforming data[5].

2.3 Splitting the dataset

After the data has been loaded in the program, I used the `sklearn.model_selection.train_test_split`[11], which returns the training set split and the testing set split of the dataset. I used a 0.8 - 0.2 training-test split and I have also set the `shuffle` parameter as `True` with a constant random state to achieve uniformity between runs of the program. Following that, I create a dictionary with two keys, `train_set`, and `test_set`. This enables me to isolate the training set from the test set to prevent data leakage[12]. At this point, we are in a position to use the training set for our data analysis and visualisation as well as the model selection process (validation process). Then, after selecting a model, we can use the test set to evaluate the selected model's final performance. This is the one and only time the testing set is ever used.

2.4 Cleaning the data

In section 2 of the paper[1] documenting the dataset used for this practical, the steps taken to prepare the data are clearly outlined. Specifically, section 2.1 describes how the element data is obtained and processed and section 2.2 describes how the data taken from the NIMS Superconducting Material Database was processed. These subsections, among other things, describe how the author dealt with missing data, seemingly misrecorded data and columns that have no value for the task at hand (such as the unique identifier of each row). Given the thorough data preparation process carried out by the author, I decided to not carry out any further data preprocessing. In order to confirm that the dataset has no missing values, I used the `pandas.DataFrame.info` method[6], which prints information about a `DataFrame`. The output of this method confirms that a `DataFrame` with 21263 entries and 82 columns is loaded into the program and all of the columns contain 21263 non-null values, exactly as described in section 2.3 of the paper[1].

3 Analysing and Visualising the data

Note: The analysis and visualisation was carried out on the training dataset only in order to prevent data leakage[12].

3.1 Feature analysis and visualisation

In order to see how each feature correlates to the critical temperature, I used the `pandas.DataFrame.corr` method[7], which uses Pearson's Correlation Coefficient[8] to measure the pair-wise statistical relationship between the features. I then took the coefficients calculated for each feature against the critical temperature and sorted them in descending order. Plotting the resulting series, I got the figure `corr.png` included in the submission in the `Images` directory. As shown by the plot, it seems that features related to valence have the largest negative correlation to critical temperature, i.e. the higher the valence the lower the critical temperature is, and (excluding the critical temperature feature) features related to a superconductors thermal conductivity seem to have the highest positive correlation to critical temperature. This plot gives a hint as to which features could be the most important in predicting a superconductor's critical temperature. To further illustrate the correlation coefficient between features I used the `seaborn.heatmap` method to create a heatmap[9], which plots the correlation matrix created by the `corr` method, as a color-encoded matrix. The heatmap was included in the submission and it is named `heatmap.png`. Using the colour scale on the right hand side of the plot, one can see that boxes with a light beige colour indicate that the corresponding pair has a high positive correlation and boxes with a dark purple colour indicate that the corresponding pair has a high negative correlation.

3.2 Distribution of critical temperature

In order to visualise the distribution of the critical temperature in the dataset, I used the `seaborn.distplot` method to plot the univariate distribution of the `critical_temp` column of the dataset. The resulting plot was included in the submission under the name `dist_plot.png` and upon examination one can quickly see that the distribution appears to be right-skewed, in that it has a long tail in the positive direction on the number line[10]. Indeed, upon examination, the median critical temperature of the dataset is 20.0K while the mean critical temperature is 34.5K.

4 Preparing the inputs and choosing a suitable subset of features

4.1 Preparing the inputs

Thus far, we have been using the `Dataframe` data structure[4] of the `pandas`[3] module. However, in order to make use of the data for Machine Learning we need to get rid of the axes labels and extract the data values only. For this reason, the `Dataframe.to_numpy` method[13] is used. Furthermore, before any Machine Learning takes place, the dataset being used is split into an array `X` containing the input features and an array `y` containing the target variables i.e. the critical temperature of the superconductors.

4.2 Choosing a suitable subset of features

The correlation matrix provides us with hints as to which features would be the most important for predicting the critical temperature of the superconductors. As a result one may argue that a good way of choosing a suitable subset of features would be to choose the N features with the N highest absolute correlation with the target variable, that is the critical temperature. In a similar vein, I incorporated the `sklearn.feature_selection.SelectKBest` class[14], called with a variable value for k, namely 20, 40, 60 and 80, which indicates the number of top features to select. The features are scored using the `sklearn.feature_selection.f_regression` method[15], which first calculates the correlation between each feature and the target variable and that correlation measure is converted to an F score, and subsequently to a p-value.

5 Selecting and training a regression model

Before diving into the specific regression models I've used for this practical, it is worth describing the several steps that are involved in selecting the appropriate models as these are steps that are carried out during the validation process of all the regression models under consideration.

5.1 Cross validation

In order to evaluate the performance of a model during the model selection process, I've incorporated the cross-validation resampling procedure[16]. During this process the training set is split into k folds (in our case $k = 10$, and for each unique group it carries out the following steps:

1. Take the group as the validation data set
2. Take the remaining 9 groups as a training data set
3. The model is fit on the training set and evaluated on the validation set
4. The evaluation score is retained and the model is discarded

At the end of the above procedure, the model's validation performance can be calculated by averaging the evaluation scores calculated at each step. I've incorporated the the k fold cross validation process by using the `sklearn.model_selection.KFold` class[17].

5.2 Grid Search

Each model will have several hyperparameters that have an impact on how well it fits the data. A way of finding the 'right' combination of hyperparameters is to carry out an exhaustive search over these parameters and evaluate the model's performance for each possible combination. In order to incorporate the above into my program in combination with the cross validation procedure, I chose to use the `sklearn.model_selection.GridSearchCV` class which accepts the `sklearn.model_selection.KFold` iterable described above. In this way, for each possible combination of hyperparameter values, the cross-validation procedure is carried out. Then, the model with the highest performance, according to a scoring measure (in our case "R squared"), can be obtained through the `best_estimator_` attribute of the `GridSearchCV` instance.

5.3 Pipeline

As mentioned in section 4.2, I use the `sklearn.feature_selection.SelectKBest` in order to choose a suitable subset of features. Ideally, we would like to choose the features that will allow the model to achieve the highest performance during validation and in turn the testing process. Since we can't know for which value of k features our regression model will achieve the highest performance, I chose to not carry out the feature selection process before the grid search, but to include it in it. For this reason it became necessary to create a pipeline, using the `sklearn.pipeline.Pipeline` class[18]. This will allow us to carry out an exhaustive search through all the possible combinations of hyperparameters as well as the different values for k.

5.4 Multiple regression model

Similarly to the paper[1], I decided to firstly use a multiple regression model. To fit this purpose, I used the `sklearn.linear_model.LinearRegression` class which fits a linear model to minimise the residual sum of squares between the predicted and the observed values of the target variable in the dataset.

5.4.1 Linear Regression hyperparameters

The only hyperparameter tuning carried out for this regression model is the `fit_intercept` hyperparameter. The two possible values for this is either `True` or `False` and it basically determines whether the intercept of this model is calculated. Simply put, it determines whether the line of best fit is forced to the origin[19] (when set to `False`).

5.4.2 Validation results

The estimator returned by the grid search process is the one with the following parameters:

- k_best features: 80
- `fitintercept: True`

With cross-validation r^2 score of **0.73**. The time elapsed for the grid search and cross-validation process was **4.75** seconds.

5.5 Ridge

A hyperparameter that we have not seen yet is regularisation. Regularisation prevents overfitting by discouraging a large change in the model's weights. In order to explore its effect on the results of our regression task, I decided to use the `sklearn.linear_model.Ridge` class[22]. This model uses the linear least squares cost function but it also applies the l2 regularisation.

5.5.1 Ridge hyperparameters

I've carried out hyperparameter tuning on two hyperparameters namely:

- `alpha` : this is the l2 regularisation value.
- `fit_intercept`: this determines whether the intercept of this model is calculated.

The values used in the grid search process were the following:

- `alpha` : $0.1, 0.5$
- `fit_intercept`: `True, False`

5.5.2 Validation results

The estimator returned by the grid search process is the one with the following parameters:

- k_best features: 80
- `alpha`: 0.1
- `fit_intercept`: `True`

With cross-validation r^2 score of **0.73**. The time elapsed for the grid search and cross-validation process was **7.09** seconds.

5.6 XGBoost

In order to compare my methodology to the one followed in the paper[1], I have decided to also use the XGBoost model. This model is described in detail in Chen and Guestrin[20], but essentially this is an advanced version of gradient boosting model, that is designed for efficiency and performance.

5.6.1 XGBoost hyperparameters

I've carried out hyperparameter tuning on several parameters namely:

- `learning_rate`: this is the step size shrinkage used in weight updates in an attempt to prevent overfitting.
- `subsample`: this is the subsample ratio of the training instances. It is the ratio of the training data that XGBoost would randomly sample prior to growing trees. This is also a means to counter overfitting. Subsampling occurs once in every boosting iteration.
- `max_depth`: this is the maximum depth of a tree. A higher value will make the model more complex and in turn more likely to overfit.

- `min_child_weight`: In linear regression tasks, this is the minimum number of instances needed to be in each node. A higher value for this parameter makes the algorithm more conservative.

Note: More details and a more exhaustive list of the parameters used by XGBoost can be found in its documentation site[21].

The values used in the grid search process were the following:

- `learning_rate`: $0.02, 0.5$
- `subsample`: $0.5, 1$
- `max_depth`: $6, 16$
- `min_child_weight`: $0.5, 1$

5.6.2 Validation results

The estimator returned by the grid search process is the one with the following parameters:

- k_best features: 80
- `learning_rate`: 0.1
- `subsample`: 0.5
- `max_depth`: 16
- `min_child_weight`: 0.5

With cross-validation r^2 score of **0.92**. The time elapsed for the grid search and cross-validation process was **1894.29** seconds.

5.7 Random Forest

Since we've used a boosting method above, one in which base estimators are sequentially built with the aim of reducing the bias of the combined estimator, it's appropriate to use the other member of the ensemble methods family, that is an averaging method. With this method, estimators are built independently and their predictions are then averaged[23]. The `sklearn.ensemble.RandomForestRegressor` regressor is one that belongs in the averaging method family, and it can be described as a meta estimator which fits a number of decision trees on various sub-samples of the dataset and then uses averaging to improve the predictive accuracy and also control over-fitting[24].

5.7.1 Random Forest hyperparameters

I've carried out hyperparameter tuning on several parameters namely:

- `n_estimators`: this is the number of trees in the forest.
- `max_features`: this is the number of features to consider when looking for the best split.
- `min_samples_split`: this is the minimum number of samples required to split an internal node.
- `min_samples_leaf`: this is the minimum number of samples required to be at a leaf node.
- `reg_bootstrap`: this indicates whether bootstrap samples are used when building trees. If `False`, the whole dataset is used to build each tree.

Note: The bootstrap method is simply a method used to estimate statistics on a population by sampling a dataset[25].
The values used in the grid search process were the following:

- `n_estimators`: $10, 100, 200$
- `max_features`: `auto, sqrt`
- `min_samples_split`: $2, 5$
- `min_samples_leaf`: $1, 2$
- `bootstrap`: `False, True`

5.7.2 Validation results

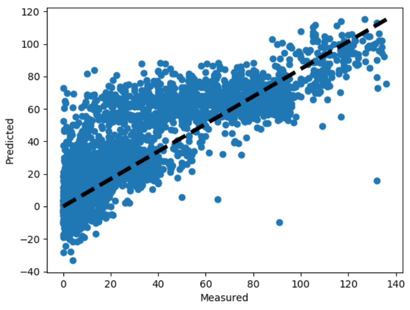
The estimator returned by the grid search process is the one with the following parameters:

- k_best features: 80
- `n_estimators`: 200
- `max_features`: `sqrt`
- `min_samples_split`: 2
- `min_samples_leaf`: 2
- `bootstrap`: `False`

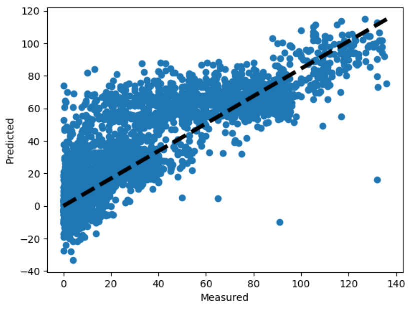
With cross-validation r^2 score of **0.93**. The time elapsed for the cross-validation process was **5544.07** seconds.

6 Evaluating the performance of the models

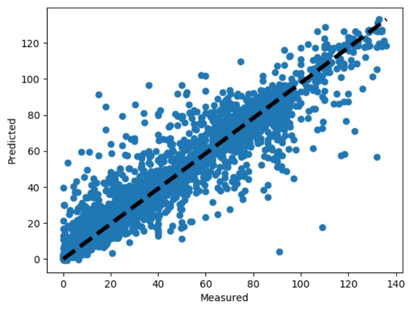
An effective comparison of the different models used to carry out the regression task would be to compare their performance on the test set, which is essentially an indication of how well they generalise on new data. Since the testing data was taken aside from the very beginning of the process, we can confidently assume that it is a good representation of the kind of data that might be introduced in the future, say when the models are deployed in production. For each of the models, a scatter plot was produced showing the predicted values against the actual values for the target variable. These are as follows:



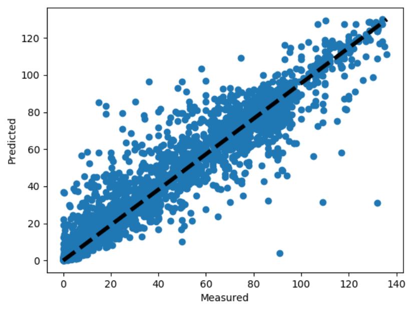
(a) LinearRegression



(b) Ridge



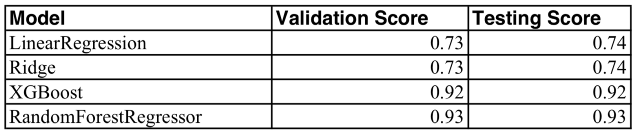
(c) XGBRegressor



(d) Random Forest Regressor

Figure 1: Predicted vs Actual values for the critical temperature

The results can be summarised by the following table:



Model	Validation Score	Testing Score
LinearRegression	0.73	0.74
Ridge	0.73	0.74
XGBoost	0.92	0.92
RandomForestRegressor	0.93	0.93

(a) Table of results

Figure 2: Table of results for the various regression models

There are a number of observations we can make from the above figures and plots:

- Clearly, the `XGBRegressor` and the `Random Forest Regressor` are superior to the other two models, scoring a test score of **0.92** and **0.93** respectively.
- The `LinearRegression` and `Ridge` regression models perform equally as well. They both have a test score of **0.74**. Unsurprisingly, the scatter plots produced for these models are very similar.
- For each of the models tested, their validation scores are pretty much equal to their testing (out-of-sample) scores.

Note: The `Results` directory in the submission contains images of the plots above, as well as text files containing the output of the program.

7 Critical discussion

7.1 Results

Several discussion points can be made about the results obtained:

- The difference in performance of the different regression models were not too surprising as the `XGBRegressor` and `RandomForestRegressor` are powerful algorithms using advanced ensemble methods and are thus more capable of fitting to a complex dataset, while the `LinearRegression` and `Ridge` models can only properly fit linear data.
- The performance score for the `LinearRegression` and `Ridge` models is the same which prompted me to think about the reasons why that is the case. Since the two models are the same with the addition of the regularisation parameter in the case of `Ridge`, we can deduce that the regularisation applied does nothing to improve the model's performance in this case. And since regularisation is used to reduce overfitting, as earlier mentioned, we can say that the `LinearRegression` model does not overfit.

- The validation and testing scores for all models are incredibly close and identical in the cases of the `XGBoost` and `RandomForestRegressor`. This shows that the way the validation process was followed made for a very good estimation of the testing data. Furthermore it points to the fact that there is no overfitting taking place whatsoever. If that were the case, the validation score would be significantly higher than the testing score.

- The `RandomForestRegressor` actually performs better than the `XGBoost` model introduced in the paper[1] that came with the practical specification, which was reported to have an out-of-sample score (testing score) of **0.92**. While this is not a huge difference as it very well could be because of the distribution of samples in the training and testing set, it shows that there is indeed room for improvement.

- There was a significant difference in the times needed by each model's grid search and cross validation process to complete. Granted, a different number of hyperparameters were fine-tuned in each case and so it'd be unfair to compare the `Ridge` model which was fit with 16 combinations of hyperparameters (including the 4 parameters for `SelectKBest` with the 192 combinations for the `RandomForestRegressor`. However, even taking this difference in account, there is a stark difference between the two simpler models with the two more complex ones. There is indeed a trade-off between algorithm complexity and the time required to fit the models, and so when choosing an appropriate model, time available should be considered.

7.2 Approach

I believe that the approach I followed and the methodology I carried out is rigorous and consistent across the experiments I ran for each regression model. There are a few ways this was achieved:

- The process of loading and splitting the dataset was identical across all program runs. This is because a random seed was set an thus the training - test split was always the same, for all regression models. This ensured that the comparison of the different models was done fairly.

- Cross validation was incorporated during the validation process. This ensured that the performance of each model's performance during this phase was estimated accurately. This is proven by the results given in the previous section, as the validation and testing results are very similar, with their difference essentially being statistically insignificant.

- A grid search across a variety of hyperparameters was carried out during the validation process as well. This ensured that the hyperparameter combination that produced the best performance was used and that we didn't end up with a scenario in which the most optimal hyperparameter values were used for one of the models and then the least optimal were used for another, something which would be unfair.

- Throughout the process, I've maintained a strict approach to viewing the dataset to prevent data leakage. I've even held off evaluating each model before I completed the validation process for each of them, as the results of that step could have influenced my future decisions.

8 Conclusion

Overall, I'm very pleased with the result of my work as I've fulfilled the requirements of the specification while carrying out a proper Machine Learning methodology, starting from the data loading and visualisation all the way through to evaluating the performance of the different regression models I've used for this task. If I had more time, I'd explore more regression algorithms in to broaden my knowledge and gain more practical experience. In conclusion, this practical was very successful in achieving its learning outcomes as it allowed us to carry out a Machine Learning project from start to finish, touching on many aspects of the subject and allowing us to familiarise ourselves with concepts introduced in the lectures as well as encouraging us to explore more advanced concepts through self-study.

References

[1] Hamidieh, K. (2018). A data-driven statistical model for predicting the critical temperature of a superconductor. Computational Materials Science, 154, 346-354. doi: 10.1016/j.commatsci.2018.07.052

[2] Superconductivity. (2020). Retrieved 3 March 2020, from http://hyperphysics.phy-astr.gsu.edu/hbase/Solids/scond.html

[3] pandas.read_csv — pandas 1.0.1 documentation. (2020). Retrieved 3 March 2020, from https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

[4] pandas.DataFrame — pandas 1.0.1 documentation. (2020). Retrieved 3 March 2020, from https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html

[5] Why and How to Use Pandas with Large Data. (2020). Retrieved 3 March 2020, from https://towardsdatascience.com/why-and-how-to-use-pandas-with-large-data-9594dda2ea4c

[6] pandas.DataFrame.info — pandas 1.0.1 documentation. (2020). Retrieved 3 March 2020, from https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.info.html

[7] pandas.DataFrame.corr — pandas 1.0.1 documentation. (2020). Retrieved 3 March 2020, from https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corr.html

[8] Pearson's Correlation Coefficient - Statistics Solutions. (2020). Retrieved 3 March 2020, from https://www.statisticssolutions.com/pearsons-correlation-coefficient/

[9] seaborn.heatmap — seaborn 0.10.0 documentation. (2020). Retrieved 4 March 2020, from https://seaborn.pydata.org/generated/seaborn.heatmap.html

[10] Skewed Distribution: Definition, Examples - Statistics How To. (2020). Retrieved 4 March 2020, from https://www.statisticshowto.datasciencecentral.com/probability-and-statistics/skewed-distribution/

[11] sklearn.model_selection.train_test_split — scikit-learn 0.22.2 documentation. (2020). Retrieved 4 March 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

[12] Brownlee, J. (2020). Data Leakage in Machine Learning. Retrieved 4 March 2020, from https://machinelearningmastery.com/data-leakage-machine-learning/

[13] pandas.DataFrame.to_numpy — pandas 1.0.1 documentation. (2020). Retrieved 9 March 2020, from https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_numpy.html

[14] sklearn.feature_selection.SelectKBest — scikit-learn 0.22.2 documentation. (2020). Retrieved 4 March 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html

[15] sklearn.feature_selection.f_regression — scikit-learn 0.22.2 documentation. (2020). Retrieved 4 March 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_regression.html

[16] Brownlee, J. (2020). A Gentle Introduction to k-fold Cross-Validation. Retrieved 4 March 2020, from https://machinelearningmastery.com/k-fold-cross-validation/

[17] sklearn.model_selection.KFold — scikit-learn 0.22.2 documentation. (2020). Retrieved 4 March 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

[18] sklearn.pipeline.Pipeline — scikit-learn 0.22.2 documentation. (2020). Retrieved 4 March 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

[19] Retrieved 4 March 2020, from https://stackoverflow.com/questions/46779605/in-the-linearregression-method-in-sklearn-what-exactly-is-the-fit-intercept-par

[20] T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system 2016. Retrieved 4 March 2020, from https://arxiv.org/pdf/1603.02754.pdf

[21] XGBoost Parameters — xgboost 1.1.0-SNAPSHOT documentation. (2020). Retrieved 9 March 2020, from https://xgboost.readthedocs.io/en/latest/parameter.html

[22] sklearn.linear_model.Ridge — scikit-learn 0.22.2 documentation. (2020). Retrieved 9 March 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

[23] 1.11. Ensemble methods — scikit-learn 0.22.2 documentation. (2020). Retrieved 10 March 2020, from https://scikit-learn.org/stable/modules/ensemble.html

[24] 3.2.4.3.2. sklearn.ensemble.RandomForestRegressor — scikit-learn 0.22.2 documentation. (2020). Retrieved 10 March 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

[25] Brownlee, J. (2020). A Gentle Introduction to the Bootstrap Method. Retrieved 10 March 2020, from https://machinelearningmastery.com/a-gentle-introduction-to-the-bootstrap-method/