Practical Machine Learning Assignment Report: Supervised Learning Methods Comparison
Random Forest & Multilayer Perceptron

Matei Bejan

December 7, 2019

Goal

My goal for this project is to compare those two regressors by predicting the Gross Domestic Product growth.

The libraries used in this project are: *numpy, pandas, scikit-learn, TensorFlow, keras, fancyimpute* and *matplotlib*.

Data set

The data set I have used for this project is the World Development Indicators published by World Bank and forwarded as a .csv file through kaggle. The data set contains 1346 social and economic indicators for 247 countries and geopolitical groups, collected over 54 years, from 1960 to 2014. Thus, it presents a total of 13583 samples.

Please note that the data set's details given above are determined after the preprocessing step, as the data requires some prior restructuring.

Preprocessing

The preprocessing of the data is divided in 3 steps as follows:

1. **Restructuring the data**

   The form of the initial data set header is [CountryName, CountryCode, IndicatorName, IndicatorCode, Year, Value]. Thus, there is a noticeable amount of unnecessary data, namely CountryCode or IndicatorCode. Moreover, its structure is not suited for modelling.

	CountryName	CountryCode	IndicatorName	IndicatorCode	Year	Value
0	Arab World	ARB	Adolescent fertility rate (births per 1,000 wo...	SP.ADO.TFRT	1960	1.335609e+02
1	Arab World	ARB	Age dependency ratio (% of working-age populat...	SP.POP.DPND	1960	8.779760e+01
2	Arab World	ARB	Age dependency ratio, old (% of working-age po...	SP.POP.DPND.OL	1960	6.634579e+00
3	Arab World	ARB	Age dependency ratio, young (% of working-age ...	SP.POP.DPND.YG	1960	8.102333e+01
4	Arab World	ARB	Arms exports (SIPRI trend indicator values)	MS.MIL.XPRT.KD	1960	3.000000e+06
...	...	...	...	...	...	...
5656453	Zimbabwe	ZWE	Time required to register property (days)	IC.PRP.DURS	2015	3.600000e+01
5656454	Zimbabwe	ZWE	Time required to start a business (days)	IC.REG.DURS	2015	9.000000e+01
5656455	Zimbabwe	ZWE	Time to prepare and pay taxes (hours)	IC.TAX.DURS	2015	2.420000e+02
5656456	Zimbabwe	ZWE	Time to resolve insolvency (years)	IC.ISV.DURS	2015	3.300000e+00
5656457	Zimbabwe	ZWE	Total tax rate (% of commercial profits)	IC.TAX.TOTL.CP.ZS	2015	3.280000e+01

5656458 rows × 6 columns

Figure 1: Initial structure of the WDI data set.

A first preprocessing step will be to remove the CountryCode and IndicatorCode columns, followed by a transposition of the feature matrix so that each sample contains the country name, all the indicators associated to said country and the year the observation was made.

It is important to understand that the GDP can be calculated as either the sum of expenditure or income indicators, as is shown here. This means that predicting the GDP for a certain year based on the same years' data is futile, as the GDP has a linear dependency in relation to certain features. In order for the regression to make sense, I have replaced each year's GDP with the consecutive year's GDP and eliminated the data for the year 2015, as it is the last observed year. With the data in this format, the models can predict next year's GDP based on the current's data.

	CountryName	Adolescent fertility rate (births per 1,000 women ages 15-19)	Age dependency ratio (% of working-age population)	Age dependency ratio, old (% of working-age population)	Age dependency ratio, young (% of working-age population)	Arms exports (SIPRI trend indicator values)	Arms imports (SIPRI trend indicator values)	Birth rate, crude (per 1,000 people)	CO2 emissions (kt)	CO2 emissions (metric tons per capita)	CO2 emissions from gaseous fuel consumption (% of total)	CO2 emissions from liquid fuel consumption (% of total)	CO2 emissions from liquid fuel consumption (kt)	CO2 emissions from solid fuel consumption (% of total)	Death rate, crude (per 1,000 people)	Fertility rate, total (births per woman)	Fixed telephone subscriptions	Fixed telephone subscriptions (per 100 people)
0	Afghanistan	145.321	81.717726	5.086254	76.631472	NaN	40000000.0	51.276	414.371	0.046068	NaN	65.486726	271.358	30.973451	32.403	7.45	7700.0	0.087755
1	Afghanistan	145.321	82.755896	5.132610	77.623286	NaN	56000000.0	51.374	491.378	0.053615	NaN	59.701493	293.360	35.820896	31.902	7.45	NaN	NaN
2	Afghanistan	145.321	83.304557	5.139519	78.165038	NaN	64000000.0	51.464	689.396	0.073781	NaN	52.659574	363.033	43.085106	31.415	7.45	NaN	NaN
3	Afghanistan	145.321	83.550740	5.111892	78.438848	NaN	40000000.0	51.544	707.731	0.074251	NaN	55.440415	392.369	37.305699	30.937	7.45	NaN	NaN
4	Afghanistan	145.321	83.734442	5.056177	78.678265	NaN	56000000.0	51.614	839.743	0.086317	NaN	56.768559	476.710	35.807860	30.464	7.45	NaN	NaN

5 rows × 1346 columns

Figure 2: Transposed indicators.

2. **Filtering features with high missing values counts**

   As figure 2 suggest, our data set contains NaNs, and a large number at that. In fact, there are 1121 columns with at lease 50% NaNs.

   A first step in my feature selection process was to drop all the columns which present more than 50% missing data. I've considered the columns with a large amount of missing values not being able to provide enough information about the value we wish to predict.

3. **Missing data imputation**

   I have further dealt with the missing data by imputing it using the *kNN* and the *SoftImpute* methods provided by the fancyimpute library. I have divided this approach in two parts.

   The first step is based on Beretta and Santaniello's insight on the matter of nearest neighbour imputation. In their paper, the authors suggest that kNN gives an adequate trade-off between the precision of imputation and the ability to preserve the nature of the data when the maximum percentage of missing values lies between 15% and 30%. Furthermore, they note that a value of $k = 3$ has given the best results in respect to the precision-preservation trade-off mentioned above [**1**].

   Respecting Beretta and Santaniello's indications, I applied 1NN on the subset of features with at most 10% NaNs and 3NN then on the subset of feature with at most 30% NaNs.

   The second imputation step consisted of applying the SoftImpute [**2**] function in order to infer the missing data of those columns that present 30% to 50% NaNs. This imputation step is preceded by a bi-scaling procedure [**3**]. Bi-scaling is equivalent to a double normalization of the data through iterative estimation of row/column means and standard deviations.

Feature selection

After the data has been filtered in respect to the NaN counts, imputed and normalized, I have conducted an automated feature extraction step. For this purpose, I have chosen to employ the Lasso regression model after consulting Tibrishani's work on this subject [**4**].

Compared to step-wise feature selection using linear regression, Lasso is more computationally expensive, but yearns better results. Lasso penalizes the l_1 norm of the weights, which forces many of them to zero, thus allowing the "most relevant" variables to have nonzero weights. In my implementation, this penalization coefficient is chosen by cross-validation.

I will extend the subject of parameter search and cross-validation in the **Training** section.

I have run Lasso cross-validated regression on a 70-30 train-test split of the processed WDI data, with 5 folds. For this end I have employed sk-learn's LassoCV, which combines the training and cross validation steps, searching for the optimal penalization cost among $1, 0.1, 0.001, 0.005, 0.0005$, with a tolerance of 0.001.

The model has yearned 98 features which are considered relevant to our goal. In order to satisfy the minimum 100 features criterion, I have compensated those features two other features which Lasso hasn't chosen. Some of the relevant features can be seen in figure 3, alongside their importance score.

Training

Both the random forest regressor and the multilayer perceptron have been trained on a 70-30 train-test split of the processed data.

The **random forest** model is a supervised learning model, suitable for both regression and classification tasks. Being an ensemble model, it is composed of multiple decision trees.

The reason for which random forest have been introduced is the tendency of decision trees to overfit once they become too deep. A random forest reduces variance by either training

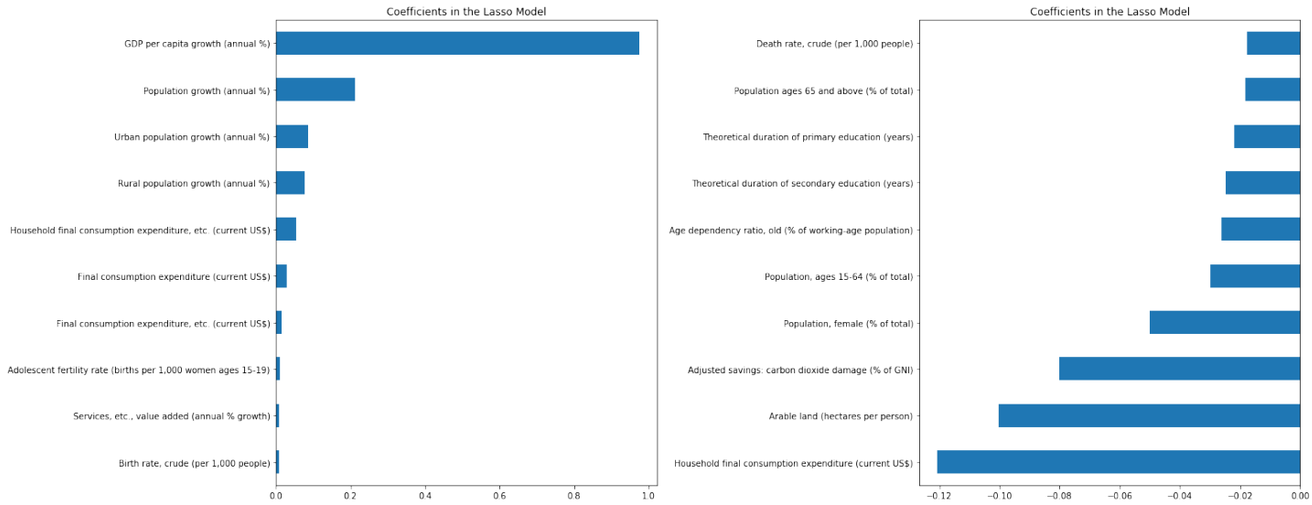

Figure 3: Top 10 indicators with highest and lowest scores, computed by Lasso.

trees on distinct data samples or by using random subsets of features. These two allow a fair use of all potentially predictive features and add a further element of randomness that prevents overfitting, respectively.

In this assignment, I have used RandomForestRegressor.

Given the forest's hyperparameters can take any values, trying different architectures by hand is impossible. For this end, I have constructed a dictionary of lists, each list containing a set of possible values.

```
parameters_cv = {
  'bootstrap': [True],
  'n_estimators': [trees for trees in range(100, 2000, 50)],
  'max_depth': [depth for depth in range(10, 300, 10)],
  'min_samples_leaf': range(2, 100, 5),
  'min_samples_split': [sample for sample in range(1, 500, 50)],
  'max_features': ['auto', 'sqrt', 'log2']
}
```

Figure 4: The random forest hyperparameter space. All values have more or less been chosen randomly, apart from bootstrap.

The parameters included in figure 4 are:

1. bootstrap: A boolean telling the random forest whether to use bootstrap in order to generate sub-samples on which the trees will be built. If *false*, the whole data set is used to build each tree.
2. n_estimators: The number of trees in the forest.
3. max_depth: Maximum depth of the trees.
4. min_samples_per_leaf: The minimum number of samples a leaf node can have.
5. min_samples_split: The minimum amount of samples considered for a split.
6. max_features: The number of features to consider when looking for the best split. If *auto*, then uses considers all features. If *sqrt* or *log*, it considers the square root or the logarithm of the number of all features, respectively.

I have then fed this list to RandomisedSearchCV, with 10 iterations and *negative mean squared error* as scoring metric. Other parameters that I have chosen are the number of concurrent workers, 16, so the search completes faster, and the level of verbosity, 10, in order to trace back any error that might come up.

Another important parameter for the RandomisedSearchCV is the cross-validation option, which I initialized with KFold with 10 folds. This algorithm consist in splitting the training data into K subsets, train the model on K-1 of them and then test it on the last subset, until all subsets have been used for testing.

Using K-Fold helps reduce both underfitting, since the whole initial training set is used for training, and overfitting, since the whole training set is used for testing.

My choice of K is based on James' and Witten's [**5**] proposal that the value of K should be either 5 or 10, since these values lead to a healthy bias-variance trade-off.

I chose randomized grid search over the exhaustive grid search because, even though the latter yearns the best set of parameters, the search space scales exponentially with more parameters or more choices for parameters adding to the grid.

Together, the two procedures produced the following set of best hyperparameters, on which I have trained the random forest model:

```
Best hyperparameters:
{'n_estimators': 100, 'min_samples_split': 51, 'min_samples_leaf': 2, 'max_features': 'auto', 'max_depth': 90, 'bootstrap': True}
```

Figure 5: Best hyperparameters for the random forest algorithm, yearned by the RandomizedSearchCV method with 10 iterations with a K-Fold cross-validation with 10 folds.

The **multilayer perceptron** model is a supervised learning model, that, just like the random forest, can be employed in both regression and classification tasks. The multilayer perceptron is a type of feed-forward neural network which consists of at least three layers: an input layer, a hidden layer and an output layer.

In this assignment I have modelled the MLP by using TensorFlow in conjunction with Keras.

Following the universal approximation theorem, I've started modelling my network with one dense layer of 32 perceptrons with ReLU as an activation function, using the Adam opimizer with a learning rate of 0.0001 and a mean squared error loss function. After training the initial network, I have added a 2^{nd} layer with 16 perceptrons and scored even better results.

After modelling 3-layer network, I've used a trial-and-error approach, manually changing the MLP's hyperparameters. I've tried different configurations, increasing the number of layers, the number of neurons per layer and using batch normalization and dropout layers. I have also tried the Adamax, SGD and RMSprop with different learning rates, between $0.001, 0.0005$ and 0.0001 and different numbers of epochs, between 10 and 30. However, all these configuration have failed to provide better results than the 3-layer network previously described.

Each network has been validated on 1681 values out of the test set.

To further improve the 30-16-1 network model, I have tweaked the number of epochs to 15 and the learning rate to 0.0003. This combination seems to yield the best results.

```
Train on 7827 samples, validate on 1681 samples
Epoch 1/15
7827/7827 [==============================] - 20s 3ms/sample - loss: 0.3828 - val_loss: 0.2581
Epoch 2/15
7827/7827 [==============================] - 20s 2ms/sample - loss: 0.1910 - val_loss: 0.1232
Epoch 3/15
7827/7827 [==============================] - 20s 3ms/sample - loss: 0.0829 - val_loss: 0.0550
Epoch 4/15
7827/7827 [==============================] - 20s 3ms/sample - loss: 0.0600 - val_loss: 0.0429
Epoch 5/15
7827/7827 [==============================] - 20s 3ms/sample - loss: 0.0493 - val_loss: 0.0403
Epoch 6/15
7827/7827 [==============================] - 20s 2ms/sample - loss: 0.0419 - val_loss: 0.0279
Epoch 7/15
7827/7827 [==============================] - 20s 3ms/sample - loss: 0.0362 - val_loss: 0.0236
Epoch 8/15
7827/7827 [==============================] - 20s 3ms/sample - loss: 0.0310 - val_loss: 0.0248
Epoch 9/15
7827/7827 [==============================] - 20s 2ms/sample - loss: 0.0262 - val_loss: 0.0291
Epoch 10/15
7827/7827 [==============================] - 20s 3ms/sample - loss: 0.0238 - val_loss: 0.0152
Epoch 11/15
7827/7827 [==============================] - 20s 3ms/sample - loss: 0.0209 - val_loss: 0.0124
Epoch 12/15
7827/7827 [==============================] - 20s 2ms/sample - loss: 0.0178 - val_loss: 0.0147
Epoch 13/15
7827/7827 [==============================] - 20s 3ms/sample - loss: 0.0157 - val_loss: 0.0180
Epoch 14/15
7827/7827 [==============================] - 20s 3ms/sample - loss: 0.0132 - val_loss: 0.0102
Epoch 15/15
7827/7827 [==============================] - 19s 2ms/sample - loss: 0.0117 - val_loss: 0.0062
<tensorflow.python.keras.callbacks.History at 0x7fde722d9e80>
```

Figure 6: The MLP's validation scores for all the training epochs.

Results Comparison

I have compared the two models through both metric computation and graphic representation of their results.

The first plot, which can be seen in figure 7, is the histograms of residuals. Residuals are a more convenient denomination for the prediction error, being computed by subtracting the predictied values from the ground truths.

Both the random forest and the MLP have a high count of residuals in $x = 0$ and around that point, indicating minimal distance between the predicted values and the ground truths.

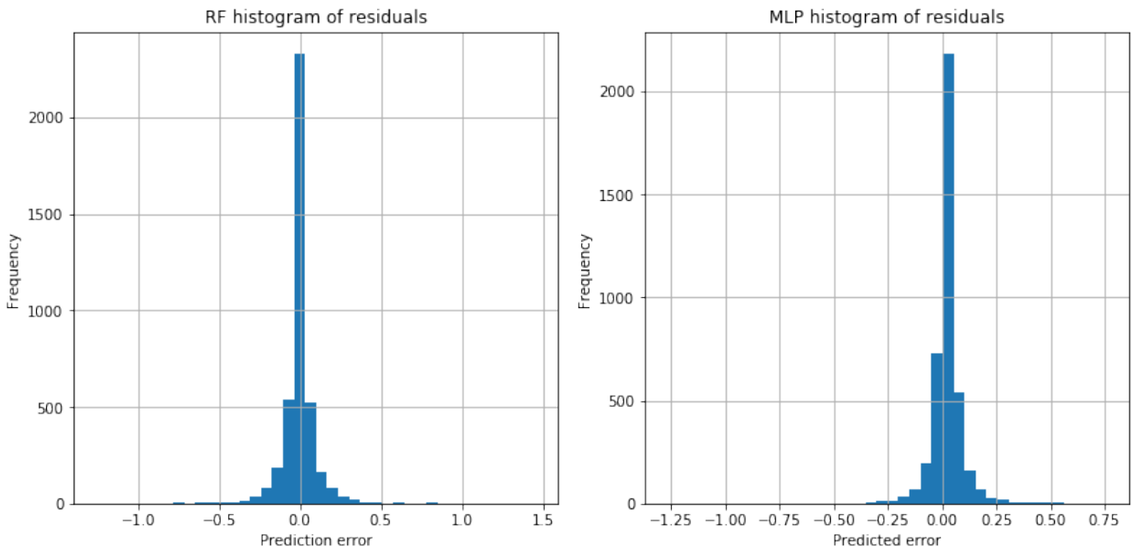

Figure 7: Side-by-side comparison of the histograms of prediction errors for random forest and MLP, respectively.

The second plot is the scatter plot of residuals. This can highlight trends in the way our models learn. A "good" residuals plot is considered to be centered around the $(0, 0)$ point and/or scattered symmetrically in respect to the $x = 0$ line.

In figure 8 one can see both the random forest's and the MLP's residuals' scatter plots. Both models' residuals are clustered around $(0, 0)$, which just like the residuals histogram, indicates a good prediction rate. One may also observe there is a larger count of outliers in the random forest's plot.

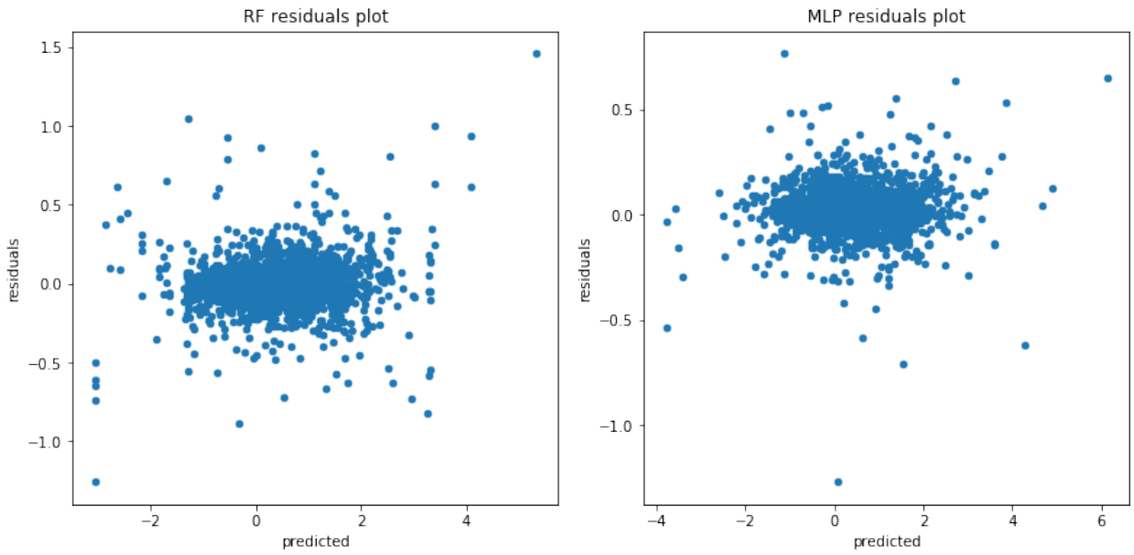

Figure 8: Side-by-side comparison of the residuals plots for random forest and MLP, respectively.

One last informative plot is the QQ-normal plot, or the quantile-quantile plot. The QQ-normal plot is determined by plotting the distribution of our residuals against a normal distribution. Its purpose is to help describe our results by comparing it with a normal distribution.

As a reference, $X \sim N(0, 1)$ has a diagonal QQ plot. The QQ, plot, however, can take many shapes, the most frequent being:

1. Right-skewed and Left-skewed plots, in which the majority of the data is located to the right or to the left of the set, respectively.
2. Light-tailed, which suggest that a high portion of our data is located in the extremities.
3. Heavy-tailed, which denotes the preponderance of extreme values.

We can see from figure 7 that both the random forest's and the MLP's QQ plots are smooth and somewhat horizontal in the centre, which indicates a portion of our data has

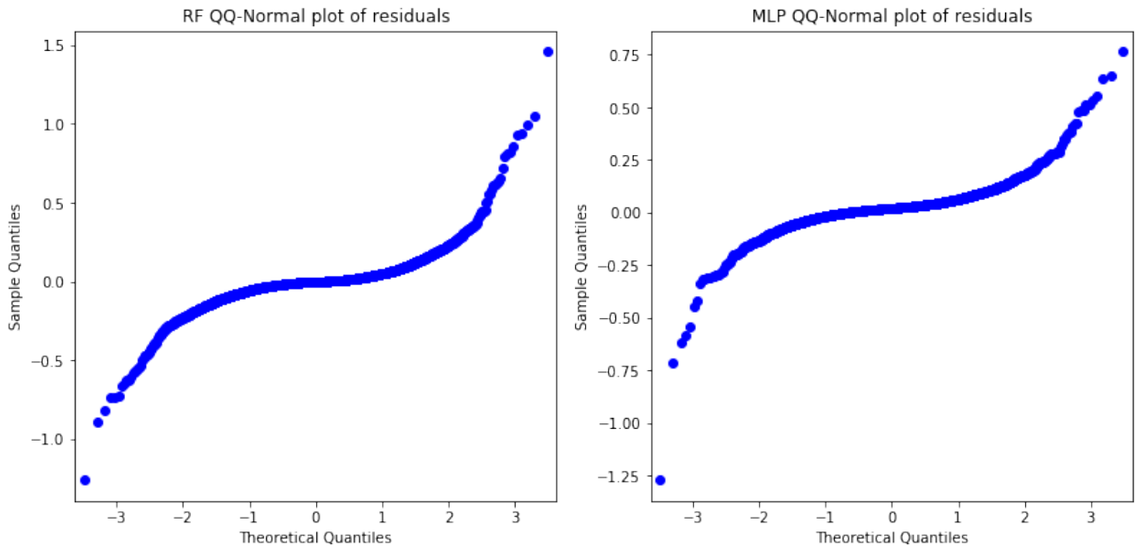

Figure 9: Side-by-side comparison of the QQ plots for random forest and MLP, respectively.

highly similar values. The plots' tails indicate the nonlinear nature of our data, whilst the isolated points to both left and right are outliers.

Below we can check the MSE, MAE and R^2 metrics computed for random forest and MLP. AS it is expected, the MLP has scored slightly better results, since it has used the MSE as its loss function. However, their R^2 metrics are extremely close. The R^2 metric illustrates the model's predictive power as a value in $[-\infty, 1]$. The closer the value is to 1, the more powerful the model. Again, we can see a slightly better score from the MLP.

```
Mean squared error for RF:0.013010657655433891 and MLP:0.0063552622739722914
Mean absolute error for RF: 0.058994634245968 and MLP:0.04830875657850179
R-square metric for RF: 0.9831729601855058 and MLP:0.9917805652759589
```

Figure 10: Computed metrics for the two models.

I have included one last plot in this report. By consulting figure 11, we can see both our models' predictive power in action. Just as the residuals and the QQ plots suggested, the predictions of both the random forest and the multilayer perceptron are accurate in respect to the ground truths.

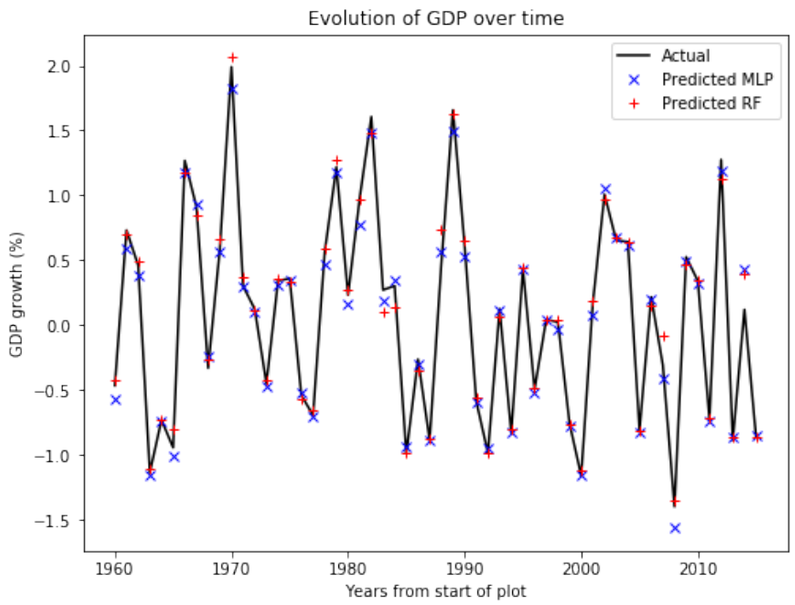

Figure 11: Sample of random forest and multilayer perceptron predictions plotted alongside the ground truths on a 54-year timeline. The points here are not necessarily drawn from the same country and have been plotted for the sake of visualizing our results.

Conclusions

In conclusion, both the random forest and the multilayer perceptron have performed well on our regression task, with the MLP scoring slightly better results.

From a modelling standpoint, each of these methods have presented pros and cons:

- Random Forest.

  The random forest regressor has been easier to model given the automatic hyperparameter search performed via random grid search and cross validation. However, this automation has come at the price of downtime. On average, the cross-validated grid search have taken 2 hours to complete. This means that trying out new restrictions for our parameters greatly increases our downtime.

- Multilayer perceptron.

  The lack of automation due to neural networks' large number of parameters has led to a trial-and-error approach on modelling, mainly based on intuition and on the universal approximation rule. Even though training one such network has taken in between 30 and 60 minutes, depending on the learning rate and the number of epochs, the total time spent testing out new parameters has been, on average, on the same scale with that necessary to train the random forest on different parameter sets.

All in all, the time spent modelling has been about the same.

One clear advantage of the random forest is the fact that parameter search and "best" model training can be easily automated, while fitting, scoring and plotting can be done overnight to minimize downtime.

On the other hand, as long as one has enough experience with neural networks, I reckon they can decide faster on where to stop adding complexity to the model.

Further improvements

An interesting aspect would be exploring the way the outliers interfere with our regressors. I hope to tend to this in my 2^{nd} assignment - on unsupervised learning methods - by introducing an anomaly detection and removal step to this existing pipeline.

References

[1] Nearest Neighbor Imputation Algorithms: A Critical Evaluation. Beretta and Santaniello, 2016.

[2] Spectral Regularization Algorithms for Learning Large Incomplete Matrices. Mazumder, Hastie and Tibshirani, 2010.

[3] Matrix Completion and Low-Rank SVD via Fast Alternating Least Squares. Trevor Hastie, Rahul Mazumder, Jason Lee and Reza Zadeh, 2014.

[4] Regression Shinkage and Selection via the Lasso. Tibshirani, 1996.

[5] An Introduction to Statistical Learning. James, Witten, Hasie and Tibshirani, 2013.