Lecture 4 Code Workshop

26 February 2024

We have reached our final live demonstration! We will be running a Random Forest (RF) on our dataset to predict the yields of reactions. It's a very similar process to Gaussian Processes and SVMs. We train (XX.fit(train data)) a model and then test it (XX.predict(test data)).

Once again, let's start by important the packages. This time around, we will use a RandomForestRegressor as our model. Simlar to the Gaussian Process models, the classifier version of the RandomForest is called the RandomForestClassifier (very creative naming). The regression metrics will again be R^2 score, MAE, and RMSE.

```
In [1]: import numpy as np
        import pandas as pd
        from sklearn.ensemble import RandomForestRegressor
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import mean_absolute_error, r2_score
        import matplotlib.pyplot as plt

        def root_mean_squared_error(true, pred):
            mean_squred_error = np.mean((true - pred)**2)
            return np.sqrt(mean_squred_error)
```

We'll load our dataset as a pandas dataframe.

```
In [2]: df = pd.read_csv('doyle.csv')
```

And we'll load in our reaction fingerprints.

```
In [3]: rxn_fps = np.load('rxn_fps_inputs.npy')
```

Unlike PCAs, SVMs, and Gaussian Processes, it's not necessary to scale our inputs for Random Forests. In this sense, they are the most "idiot proof" of the models we've seen so far.

Our labels, like last time, are the yield percentage values.

```
In [5]: labels = np.array(df['yield'])
```

We define our random forest. We can also specify how many trees we want in our forest. We'll start with 2 (not really a forest, more of a deforested forest) and we'll see how quickly we can get good accuracy.

```
In [4]: rf = RandomForestRegressor(n_estimators=2)
```

We will then split our data into training and testing sets as before.

```
In [6]: train_inputs, test_inputs, train_labels, test_labels = train_test_split(rxn_fps, labels,
                                                                                test_size = 0.33,
                                                                                random_state=12)
```

And train our RF.

```
In [7]: rf.fit(train_inputs, train_labels)
Out[7]: RandomForestRegressor(n_estimators=2)
```

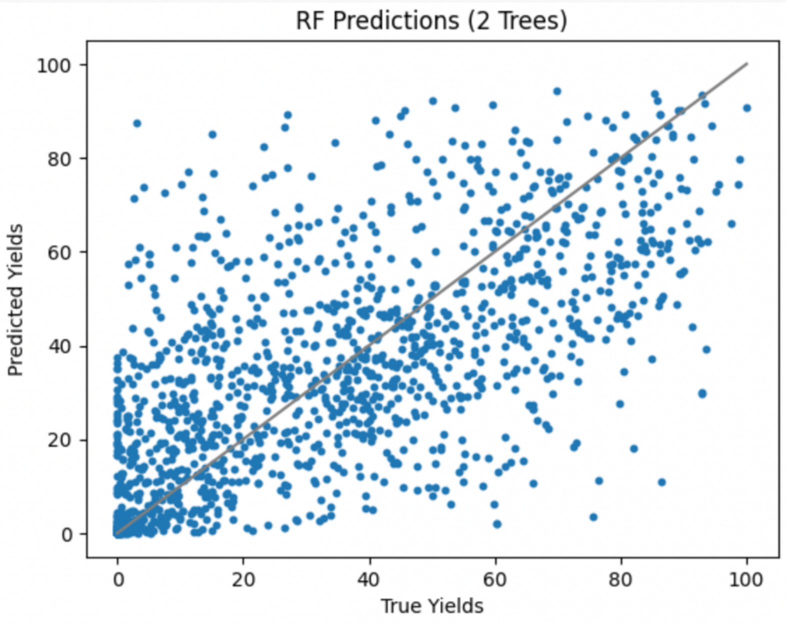
We can then predict on our held out test data.

```
In [9]: preds = rf.predict(test_inputs)
        print("MAE:", mean_absolute_error(test_labels, preds), "RMSE:", root_mean_squared_error(test_labels,
                                                                                                  preds),
             "R2:", r2_score(test_labels, preds))

        MAE: 14.285283063831026 RMSE: 20.158044273452525 R2: 0.465324306067475 66
```

Again we can plot these predictions to get a visual sense of how well we're doing.

```
In [10]: plt.scatter(test_labels, preds, s=10)
         plt.plot([0,100],[0,100], c='gray')
         plt.title('RF Predictions (2 Trees)')
         plt.xlabel('True Yields')
         plt.ylabel('Predicted Yields')
```

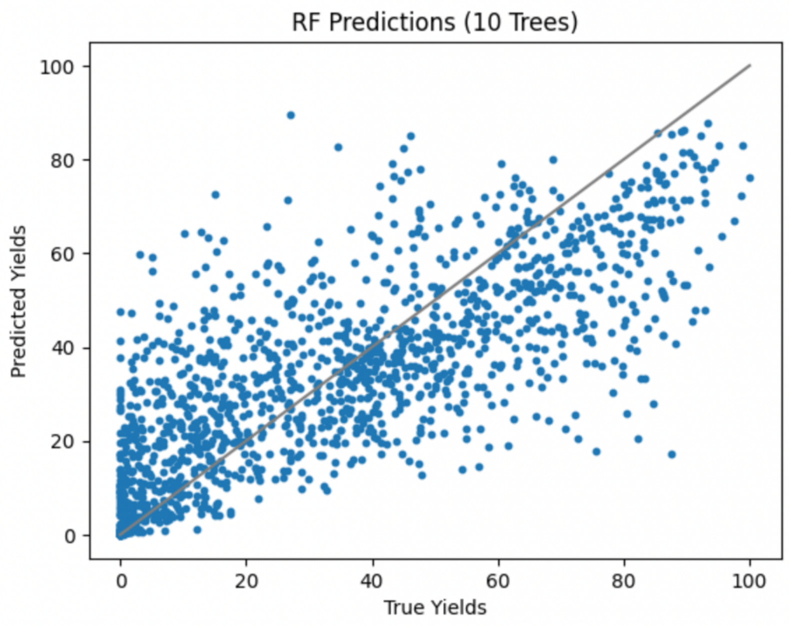


What if we increase to 10 trees in our forest? Well, we can define our new RF and re-run the training and predictions. Note that we are using the same test set so as to have an apples to apples comparison.

```
In [11]: rf = RandomForestRegressor(n_estimators=10)
         rf.fit(train_inputs, train_labels)
         preds = rf.predict(test_inputs)
         print("MAE:", mean_absolute_error(test_labels, preds), "RMSE:", root_mean_squared_error(test_labels,
                                                                                                   preds),
              "R2:", r2_score(test_labels, preds))

         MAE: 12.25451479790481 RMSE: 16.96950840227857 R2: 0.6210933959481579
```

```
In [12]: plt.scatter(test_labels, preds, s=10)
         plt.plot([0,100],[0,100], c='gray')
         plt.title('RF Predictions (10 Trees)')
         plt.xlabel('True Yields')
         plt.ylabel('Predicted Yields')
```



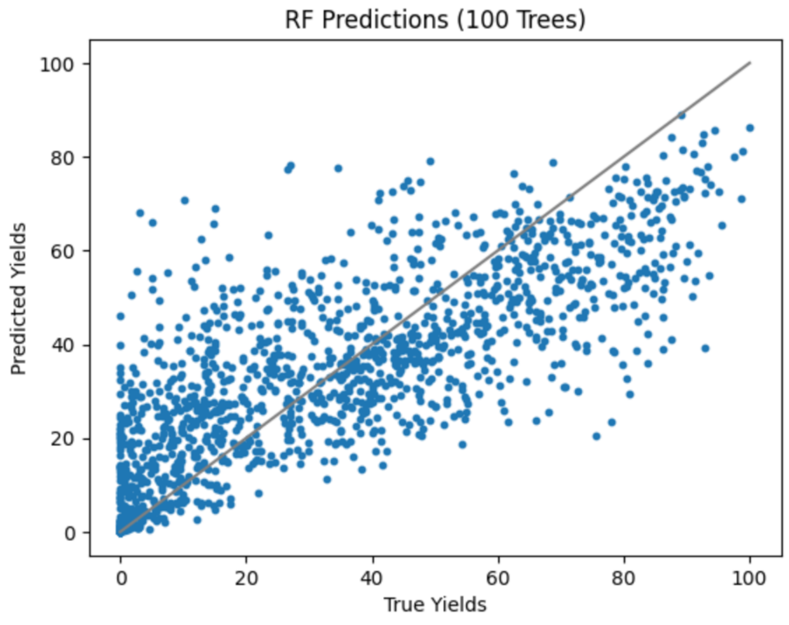
Kind of incredible what just 10 trees can do for you.

Okay, what about 100 trees? This is actually the new default option in scikit.

```
In [13]: rf = RandomForestRegressor(n_estimators=100)
         rf.fit(train_inputs, train_labels)
         preds = rf.predict(test_inputs)
         print("MAE:", mean_absolute_error(test_labels, preds), "RMSE:", root_mean_squared_error(test_labels,
                                                                                                  preds),
               "R2:", r2_score(test_labels, preds))

         MAE: 11.987471793217269 RMSE: 16.354215087248562 R2: 0.6480726148027205
```

```
In [14]: plt.scatter(test_labels, preds, s=10)
         plt.plot([0,100],[0,100], c='gray')
         plt.title('RF Predictions (100 Trees)')
         plt.xlabel('True Yields')
         plt.ylabel('Predicted Yields')
```



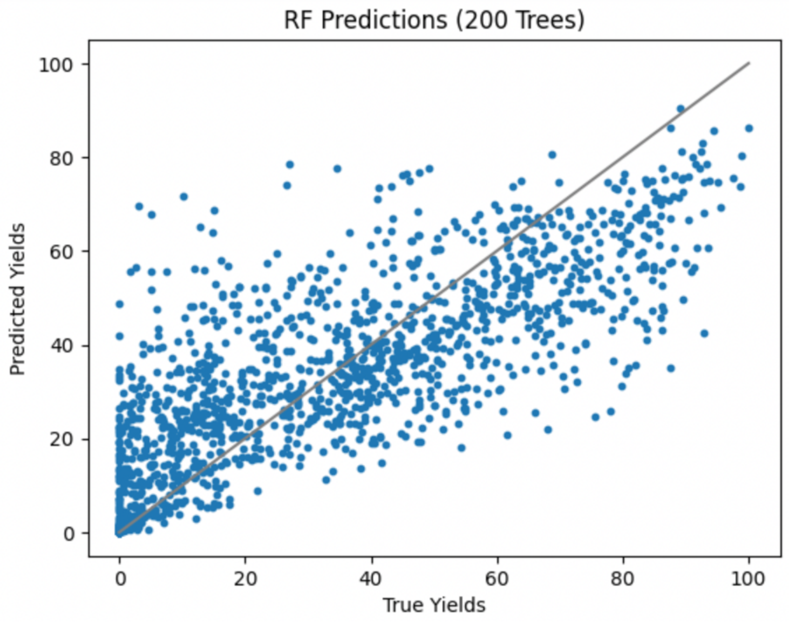
We can see a noticeable improvement, but definitely not as drastic as going from 2 to 10 trees.

How about 200 trees?

```
In [15]: rf = RandomForestRegressor(n_estimators=200)
         rf.fit(train_inputs, train_labels)
         preds = rf.predict(test_inputs)
         print("MAE:", mean_absolute_error(test_labels, preds), "RMSE:", root_mean_squared_error(test_labels,
                                                                                                  preds),
              "R2:", r2_score(test_labels, preds))
```

```
MAE: 11.901714814676033 RMSE: 16.274115675208485 R2: 0.6515115012594233
```

```
In [16]: plt.scatter(test_labels, preds, s=10)
         plt.plot([0,100],[0,100], c='gray')
         plt.title('RF Predictions (200 Trees)')
         plt.xlabel('True Yields')
         plt.ylabel('Predicted Yields')
```



You may start to notice that a RF with this many trees does start to slow you down a bit and to top it off, we really aren't seeing noticeable improvement anymore (although more trees doesn't hurt).

And that's the power of RFs! Really fast, generally pretty accurate, and no preprocessing is needed.

– Happy Coding!