Random Forest (Bagging)

Machine Learning II

December 23, 2020

1 Description

Random Forest is an implementation of Decision Trees in which we use *bagging*. Bagging stands for *Bootstrap aggregating* and it consists of training several predictors (trees) with the condition that each predictor is trained with a subsample of the total training data.

Finally, after we have the predictions, we average them (regression) or use the majority vote (classification)

2 Key concepts

- Ensemble: using several models and average their predictions
- Bagging: Ensemble of models training on subsamples of the data
- Random Forest: Ensemble using Decision Trees and Bagging
- Variable Importance: how useful is a certain variable in the performance of the model

3 How it works?

Random Forests use Decision Trees and a special mode of *bagging* to make predictions.

This is the bagging algorithm explained for M models, trained with data (X, Y) consisting of N observations and P features.

```
# bagging
1. Choose the n observations (n > N) to train each model
2. Take a random sample with replacement of n records: X,Y -> X_m, Y_m
3. Train a model with X_m and Y_m and make prediction f_m
4. Increment m: m = m + 1
5. If m <= M go to 1
6. Average all the predictions f = (f_1 + f_2 + ... + f_m ) / M
```

Now that we now how bagging works, we can create the Random Forest algorithm

```
# random forest
1. take a random sample of n observations (n < N) with replacement and p
   features (p < P) without replacement
2. apply partitioning to the sample (n, p)
3. select the feature and split_value that maximizes homogeneity
4. proceed to the next split and repeat 2.
5. continue splitting until tree is fully grown
6. back to 1 to take a new random sample and train another tree
```

With our Random Forest up and running we can not only make predictions, but also understand better how the model works according to the different features by using the Feature Importance metric. This metric can be obtained with scikit-learn with the attribute `feature_importances_` of our Random Forest.

Feature importance refers to how much each single feature has contributed to reduce the impurity when applied in the partitioning. The higher the importance, the more useful a feature is to create predictions.

4 What could go wrong?

Just like in Decision Trees, and given that Random Forests rely on Decision Trees as their fundamental pieces, we need to tackle the fact that trees will grow to a point where we can fall into overfitting. Even though that when we apply Random Forest we are sort of using some regularization by not selecting all the features in each tree (Lasso-like).

Again, we can control the growth of each inidcidual tree with the following hyperparameters

- `max_depth`: maximum depth of the tree. It can be visualized as the maximum length between the root node to a leaf.
- `min_sample_split`: minimum number of observations that must be present in a partition to be split in a node.
- `min_samples_leaf`: minimum number of observations that must be contained in a leaf (terminal partition).

Plus, for Random Forest specifically we can also tune the following to avoid overfitting:

- `max_features`: number of features (p) to pass to each tree to be trained on. Default value is $\sqrt{P}$
- `max_samples`: number of observations (n) to pass to each tree to be trained on. Default is N

5 When to use it?

- It's a robust estimator for both classification and regression jobs
- Not as good in model explainability as Decision Trees but its Feature Importance attribute makes it quite good
- Random Forests can deal with almost everything, making them very stable and not very sensitive to different scales, categorical features, etc.

6 Summary

Type	Regression, Classification
Tolerates many features?	Strong
Parametrization	Simple, intuitive
Memory needed	Very Large
Data requirements	Large
Overfitting	Medium
Difficulty	Average
Training time	Costly
Prediction time	Costly
Interpretability	Good