HyperionDev

Random Forests

September 2024

Data Science Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

Data Science Session Housekeeping cont.

- For all **non-academic questions**, please submit a query: www.hyperiondev.com/support

- Report a **safeguarding** incident: www.hyperiondev.com/safeguardreporting

- We would love your **feedback** on lectures: **Feedback on Lectures**

Learning Objectives

By the end of this lesson, learners should be able to:

- ❖ Understand fundamental concepts and advantages of **ensemble methods** to improve predictive performance and robustness of decision trees.
- ❖ Describe **bootstrapping** and its role in creating diverse subsets of training data for ensemble methods.
- ❖ Explain **bagging** (bootstrap aggregation) technique, highlighting how it reduces variance and improves the stability of predictions.

HyperionDev

Learning Objectives

❖ Identify the key difference between **random forests** and other **ensemble** methods, emphasizing **feature randomness**

❖ Interpret **feature importance** in random forests and determine the **most influential variables** for prediction.

Learning Objectives

- ❖ Apply **random forests** using Python libraries like *scikit-learn*, training models on **real-world datasets,** and comparing performances with individual decision trees and other ensemble methods.

- ❖ Experiment with **hyperparameter tuning** for random forests (n_estimators, max_depth), to **optimise model performance** and understand the impact of these parameters on the **bias-variance trade-off**.

Ensemble Methods

Introduction

HyperionDev

Ensemble Methods

- **Decision Trees** are easy to understand, apply, interpret and visualise. However, they are **not very robust**, **small perturbations in the training data** could give rise to **substantially different predictions** at test time.

- Predictions of decision trees have very **high variance.** Ideally, we'd like our models to capture general patterns, not to be **so dependent on the data** they have trained on that a bit of noise or a different sample changes predictions entirely.

Ensemble Methods

- **Ensemble techniques** work like a group of diverse experts teaming up to make decisions, and create a more robust solution than any individual could achieve alone.

- Ensemble methods **aggregate the predictions of multiple classifiers/regressors** into a single, improved prediction.

- Aside from **Random Forests**, **ensemble methods** can and do get applied to methods **other than decision trees,** but trees can benefit in particular due to how flexible they are.

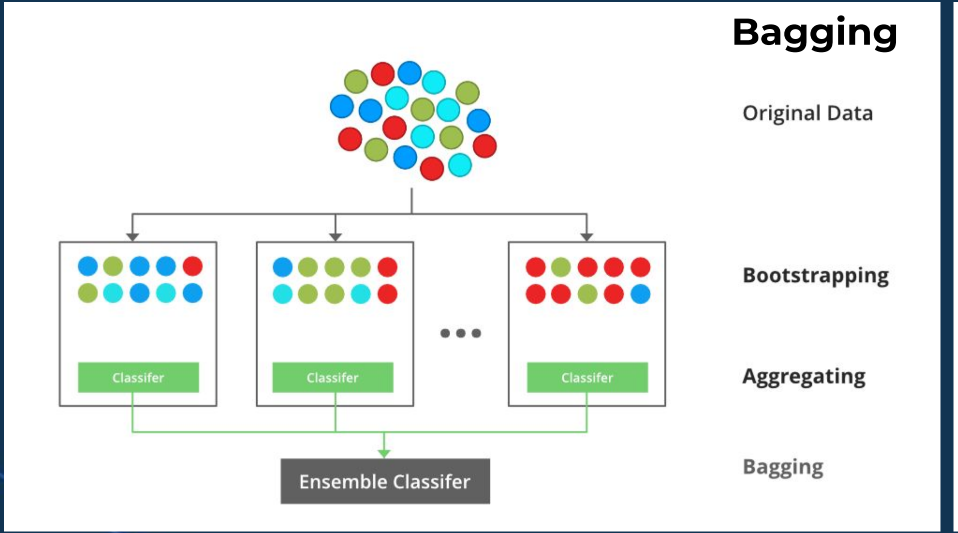
HyperionDev

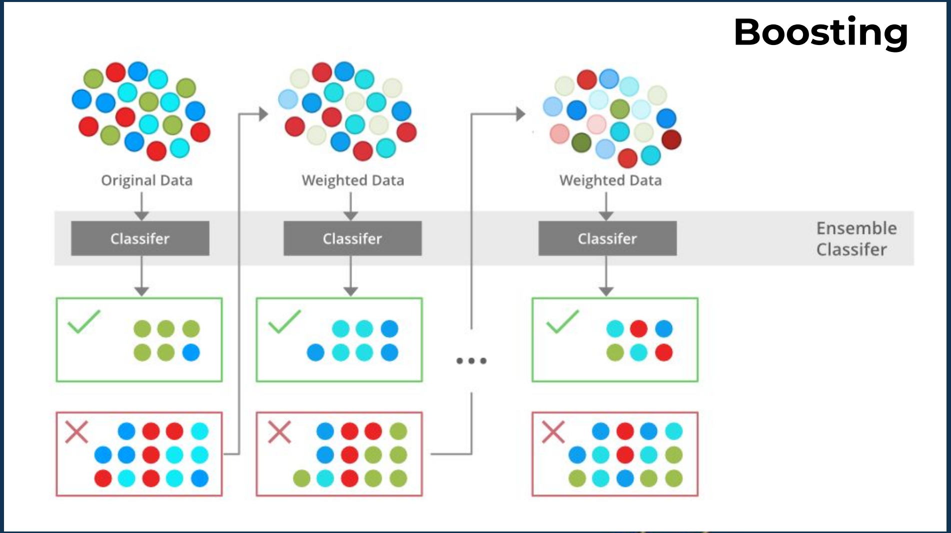
Ensemble Methods

Bagging: Bootstrap aggregation	Boosting
Trains multiple weak models in **parallel** on different subsets of the training data.	Trains multiple based models **sequentially**.
Each model is built independently.	New models are influenced by the performance of previously built models.
Training data subsets are selected using row sampling prediction is made by **averaging predictions** (regression) and **majority vote** (classification).	Each model tries to **correct the errors** made by the previous models and is **trained on a modified version of the dataset.**
Aim to **decrease variance**, solve the over-fitting problem, use for unstable models.	Aim to **decrease bias**, use for stable but simple models.
Each model receives **equal weight.**	Models **weighted** as per **performance**.

HyperionDev

Ensemble Methods





Random Forests

Gradient Boosting, XGBoost, AdaBoost

HyperionDev

https://www.geeksforgeeks.org/bagging-vs-boosting-in-machine-learning/

Random Forests

Bootstrapping and Bagging

HyperionDev

Random Forests

- **Random Forests:** created from **many decision trees** during training phase using **random subset of the dataset** to measure a **random subset of features** in each partition.

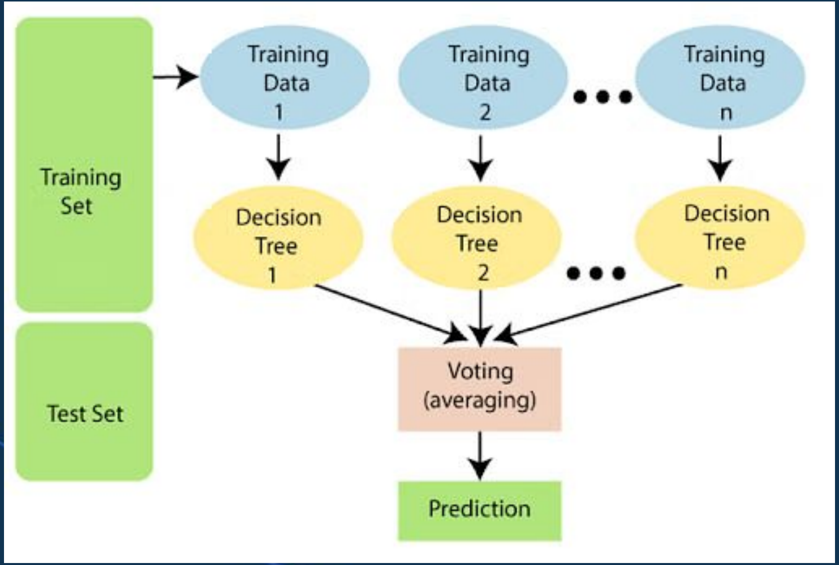
- Randomness introduces **variability** among individual trees, **reducing** the risk of **overfitting** and **improving** overall **prediction performance**.

- Aggregates predictions of all trees, either by **voting** (**classification problems**) or by **averaging** (**regression problems**).

- **Collaborative decision-making process**, supported by **multiple trees** with their insights, gives **stable** and **precise results.**

HyperionDev

Random Forests



- ❖ **Ensemble of Decision Trees,** each operates independently, minimizing the risk of the model being overly influenced by a single tree.
- ❖ **Random Feature Selection** during each tree's training, randomness ensures focus on different aspects.
- ❖ **Bootstrap Aggregating or Bagging**
- ❖ **Decision Making and Voting:** Final prediction is the **majority voting** across all trees (classification) and **average** of individual predictions (regression).

HyperionDev

Bagging (Bootstrap Aggregation)

Ensemble technique in the **Random Forest** algorithm.

- **Selection of Subset:** Choose a random sample (size n), or subset, from entire dataset (size N, n < N).
- **Bootstrapping (Bootstrap row Sampling) with replacement:** Each model is then created from these samples (Bootstrap), which are taken from the original data with replacement (instance can occur in more than one sample).
- **Independent Model Training:** Each model is trained independently on its corresponding Bootstrap Sample, generating results for each model.
- **Aggregation:** Combine all the results and generate final output based on majority voting/averaging.

HyperionDev

Key features of Random Forests

Differences with other models

HyperionDev

Key Features of Random Forests

- **Diversity:** Not all attributes/variables/features are considered while making an individual tree; each tree is different.
- **Dimensionality reduction:** Feature space is reduced.
- **Parallelization:** Each tree is created independently out of different data and attributes, fully use the CPU to build random forests.
- **Train-Test split:** No train and test data splitting required as there is always 30% of data which is not seen by the decision tree.
- **Stability:** Stable as final results are based on majority voting/ averaging.

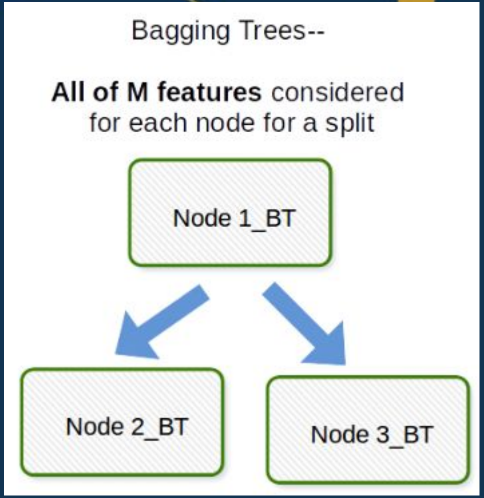
Differences with Decision Trees

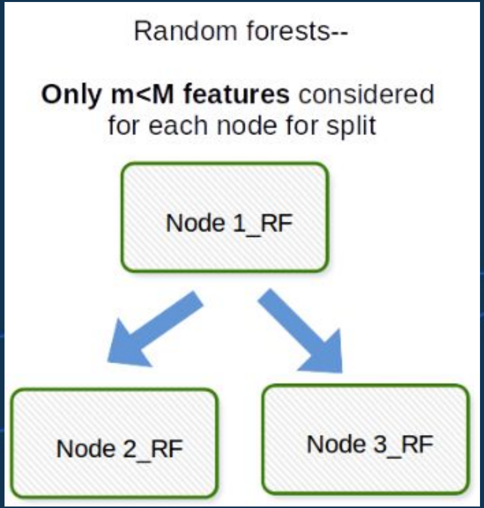
Decision Trees	Random Forests
Can suffer from **overfitting** if allowed to grow without any control.	Created from subsets of data, and final output is based on average or majority ranking; **overfitting is mitigated.** Better **bias-variance trade-off.**
When a data set with features is taken as input by a decision tree, some **rules formulated to make predictions.**	**Randomly** selects observations, builds a decision tree, and takes the average result. Does not use any set of rules.
A single decision tree is **faster** in computation.	Comparatively **slower**.

HyperionDev

Differences with Bagged Trees

Bagged Trees	Random Forests
All features are selected.	**Randomly selected features.**
Highly correlated trees, can **reduce diversity** of model. More prone to **overfitting**.	**Randomness lowers correlation** between trees, results in **diverse set** of trees, **improving** model **accuracy** by **reducing overfitting** and increasing the **diversity** of the model.
Less computationally expensive.	More computationally expensive.





$m = \sqrt{M}$ is a good place to start

Feature Selection

Feature importance

HyperionDev

Feature importance

- **Feature importance** calculates a **score** for all the input features for a given model to establish the "**importance**" of each feature in the decision-making process. The **higher the score for a feature**, the **larger effect** it has on the model to predict a certain variable.

- Note: However, **bias** is a common problem in RF models, incorrect conclusions about the importance of features.
  - Algorithm uses gain in impurity reduction as proxy for feature importance.
  - A feature with more unique values, gain in impurity reduction is artificially inflated as the model is able to split on the feature more often.
  - Model tends to *overestimate importance of features with a high number of unique values*. So check if **feature** with many unique values is **relevant** to the model or not.

HyperionDev

Feature importance

Understanding feature importance offers several advantages

- ❖ **Enhanced Model Performance:** By identifying the most influential features, you can **prioritize** them during model training, leading to more **accurate predictions**.
- ❖ **Faster Training Times:** Focusing on the most relevant features streamlines the training process, saving valuable time and computational resources.
- ❖ **Reduced Overfitting:** Overfitting occurs when a model memorizes the training data instead of learning general patterns. By focusing on important features, you can prevent the model from becoming overly reliant on specific data points.

HyperionDev

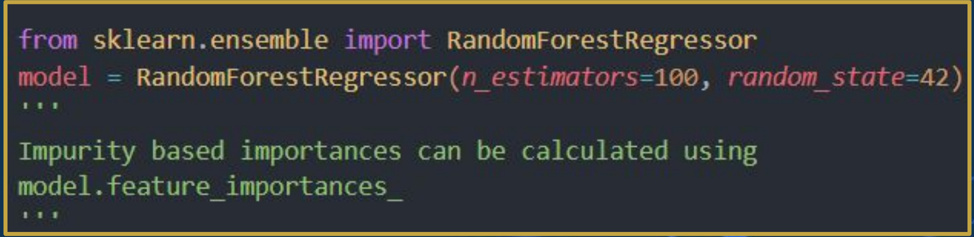
Feature Importance Methods

Random Forest Built-in Gini Importance: node impurity reduction, weighted by the number of samples that are reaching that node from the total number of samples.

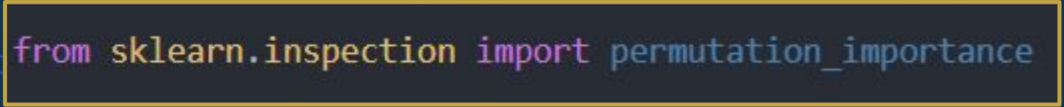
- ❖ More prone to **bias**, can inflate importance of numerical features
- ❖ Computed on statistics derived from the **training dataset,** the importances can be high even for features that are not predictive of the target variable.

Permutation Based: Calculated based on change in mean squared error (MSE) while permuting values of a feature. If permuting the values causes a huge change in the error, it means the feature is important for our model.

- ➢ Model agnostic, simple maths.



```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators=100, random_state=42)
'''
Impurity based importances can be calculated using
model.feature_importances_
'''
```



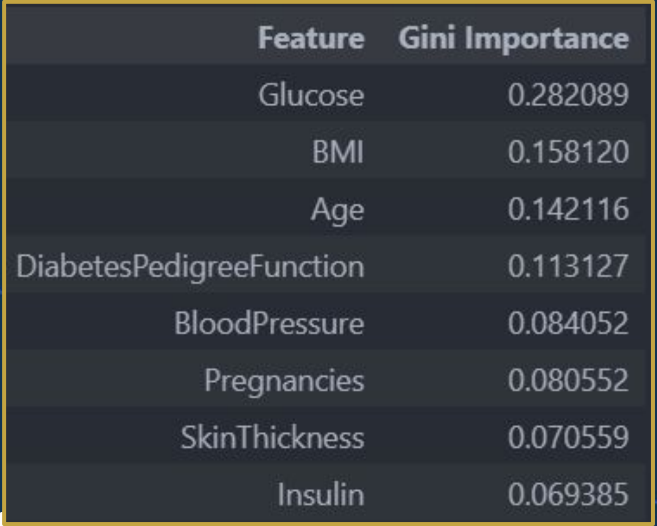
```
from sklearn.inspection import permutation_importance
```

HyperionDev

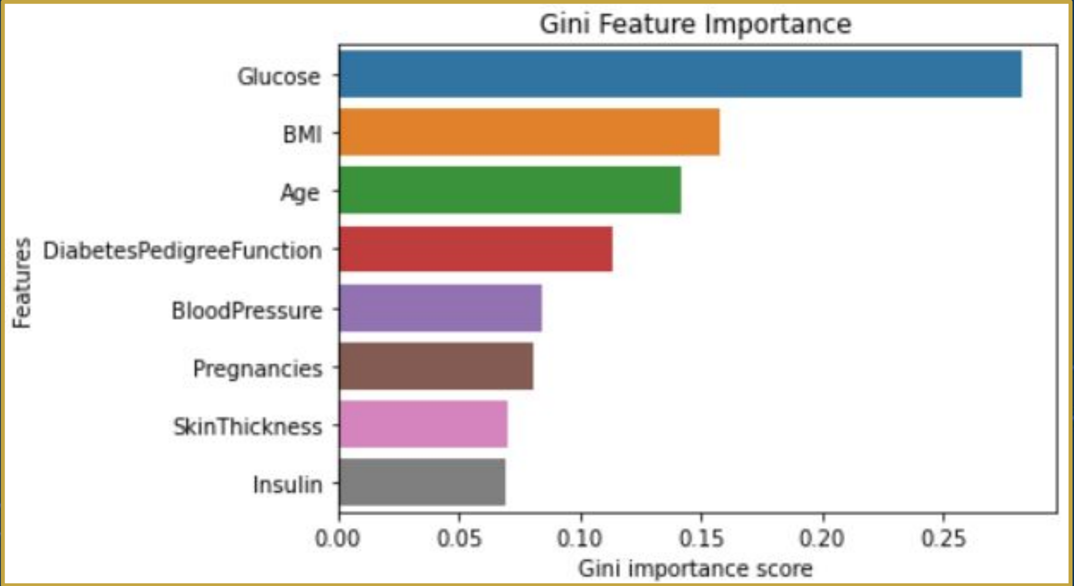
Feature importance

Built-in Gini importance

```python
#Finding the important features using the built-in Gini importance
importances = rf.feature_importances_
feature_imp_df = pd.DataFrame({'Feature': feature_names, 'Gini Importance': importances}).sort_values('Gini Importance', ascending=False)
feature_imp_df
```



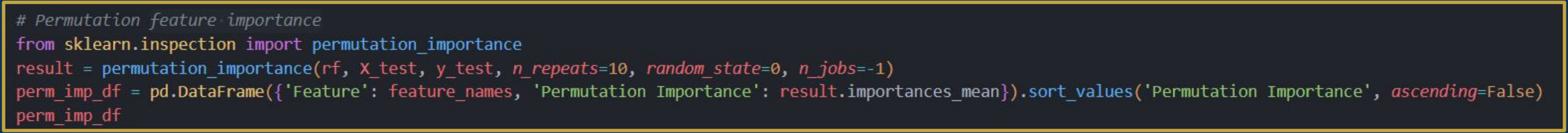
Feature	Gini Importance
Glucose	0.282089
BMI	0.158120
Age	0.142116
DiabetesPedigreeFunction	0.113127
BloodPressure	0.084052
Pregnancies	0.080552
SkinThickness	0.070559
Insulin	0.069385



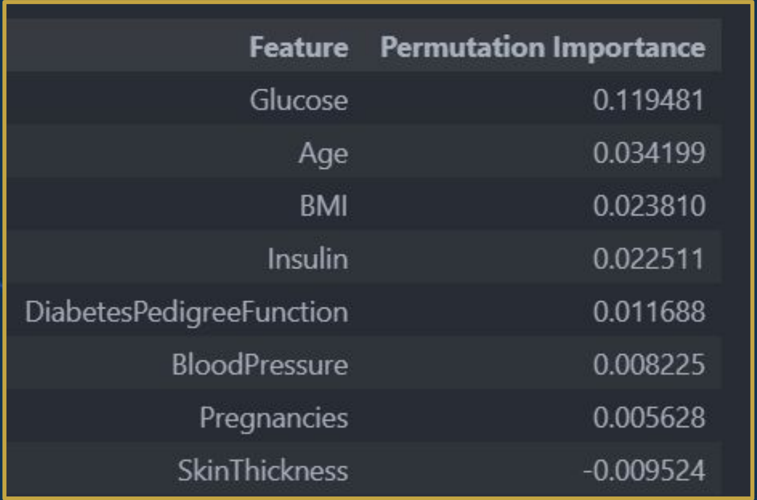
HyperionDev

Feature importance

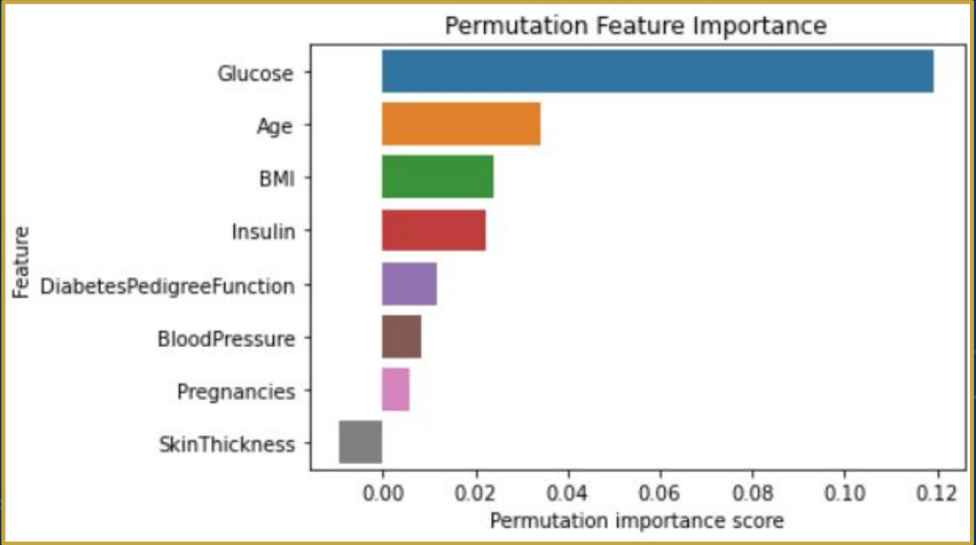
Permutation feature importance



```
# Permutation feature importance
from sklearn.inspection import permutation_importance
result = permutation_importance(rf, X_test, y_test, n_repeats=10, random_state=0, n_jobs=-1)
perm_imp_df = pd.DataFrame({'Feature': feature_names, 'Permutation Importance': result.importances_mean}).sort_values('Permutation Importance', ascending=False)
perm_imp_df
```



Feature	Permutation Importance
Glucose	0.119481
Age	0.034199
BMI	0.023810
Insulin	0.022511
DiabetesPedigreeFunction	0.011688
BloodPressure	0.008225
Pregnancies	0.005628
SkinThickness	-0.009524

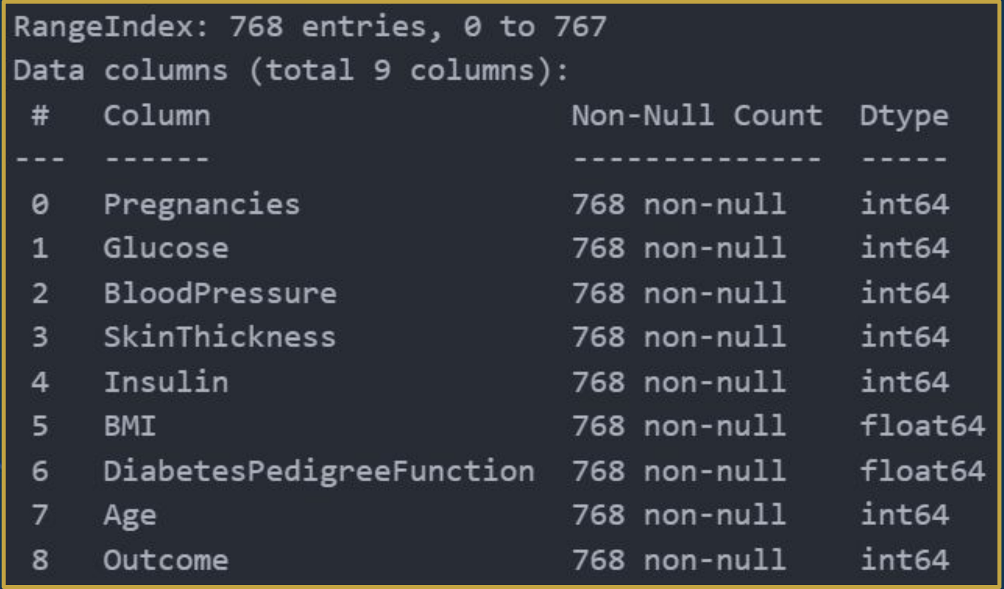


HyperionDev

Implementing Random Forests

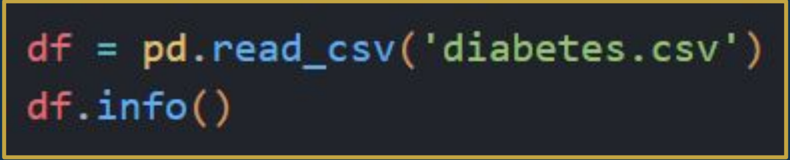
HyperionDev

Random Forests with Diabetes dataset

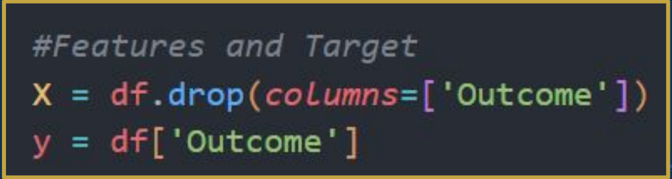


```
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
```

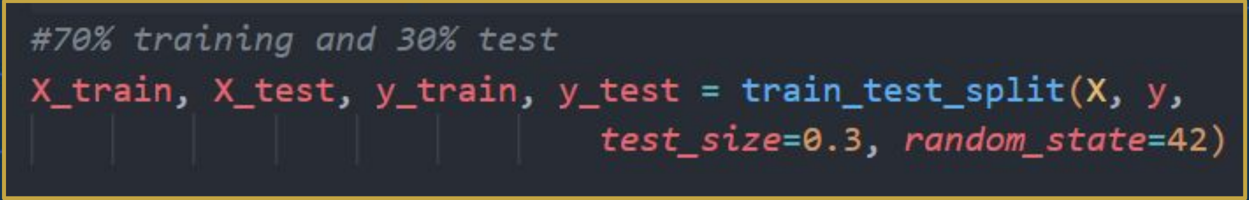
Classify and predict diabetes based on **features**.
Target is Outcome = 0 for not diabetic and 1 for diabetic.



```python
df = pd.read_csv('diabetes.csv')
df.info()
```



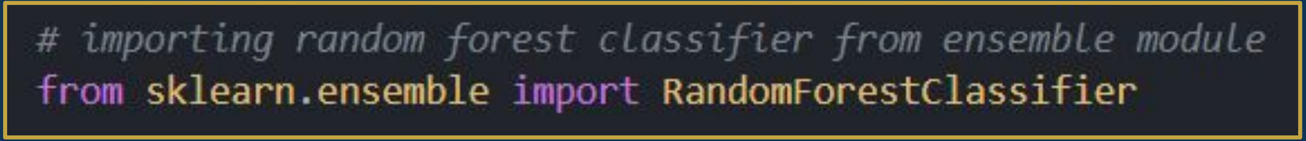
```python
#Features and Target
X = df.drop(columns=['Outcome'])
y = df['Outcome']
```



```python
#70% training and 30% test
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3, random_state=42)
```

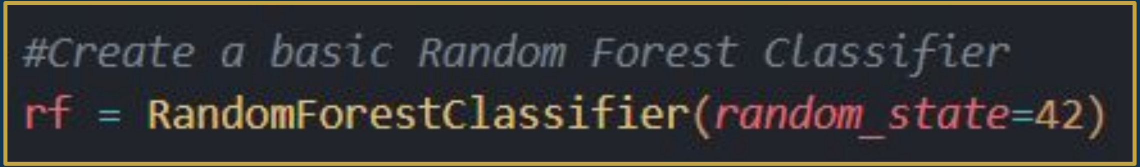
HyperionDev

Implementing Random Forests

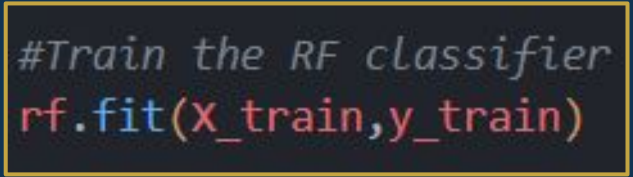


```python
# importing random forest classifier from ensemble module
from sklearn.ensemble import RandomForestClassifier
```

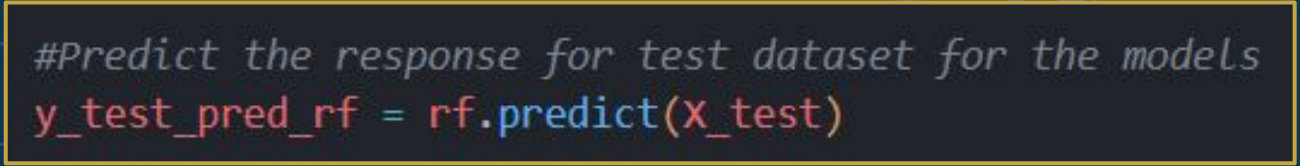
`RandomForestRegressor` for Regression models (will see an example in Tutorial)



```python
#Create a basic Random Forest Classifier
rf = RandomForestClassifier(random_state=42)
```



```python
#Train the RF classifier
rf.fit(X_train,y_train)
```



```python
#Predict the response for test dataset for the models
y_test_pred_rf = rf.predict(X_test)
```

HyperionDev

Hyperparameter Tuning

Optimisation and Bias-Variance Trade Off

HyperionDev

Hyperparameter Tuning

Hyperparameter to increase the Predictive Power

❖ **n_estimators:** number of trees the algorithm builds before taking the maximum voting or taking the averages of predictions. In general, a higher number of trees increases the performance and makes the predictions more stable, but it also slows down the computation.

❖ **max_features**: maximum number of features RF considers to split a node.

❖ **min_sample_leaf**: minimum number of leafs to split an internal node.

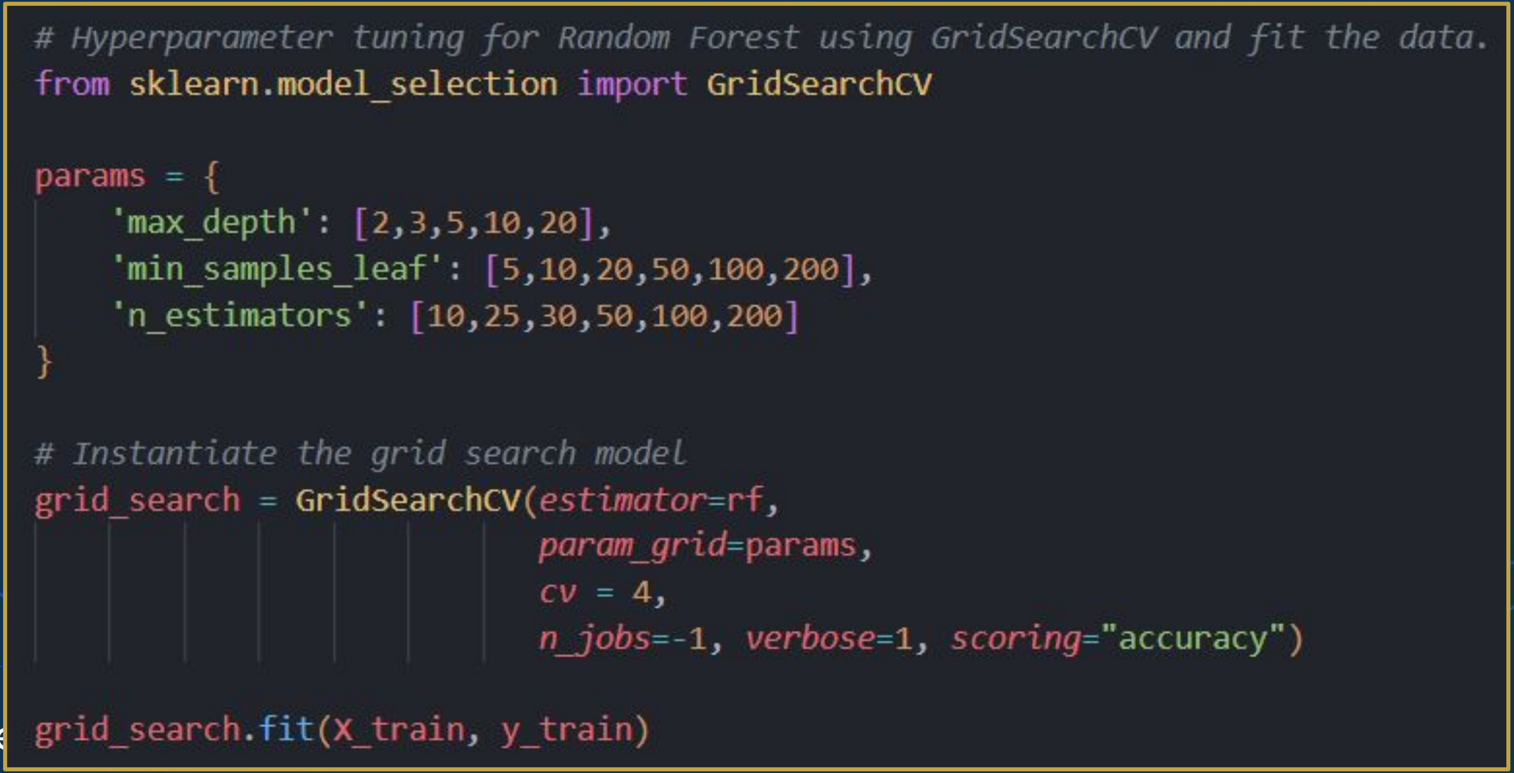
Hyperparameters to increase the RF model's speed
n_jobs, random_stat, oob_score (out-of-bag sampling).

HyperionDev

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Hyperparameter Tuning

Exhaustive search over specified parameter values for an estimator

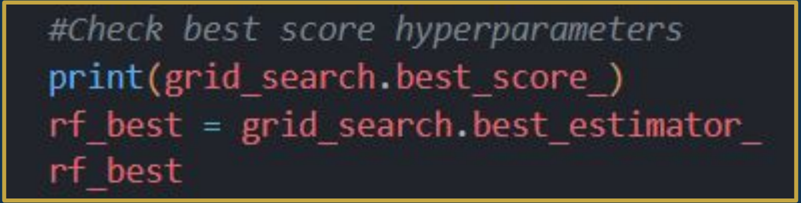


```
# Hyperparameter tuning for Random Forest using GridSearchCV and fit the data.
from sklearn.model_selection import GridSearchCV

params = {
    'max_depth': [2,3,5,10,20],
    'min_samples_leaf': [5,10,20,50,100,200],
    'n_estimators': [10,25,30,50,100,200]
}

# Instantiate the grid search model
grid_search = GridSearchCV(estimator=rf,
                           param_grid=params,
                           cv = 4,
                           n_jobs=-1, verbose=1, scoring="accuracy")

grid_search.fit(X_train, y_train)
```

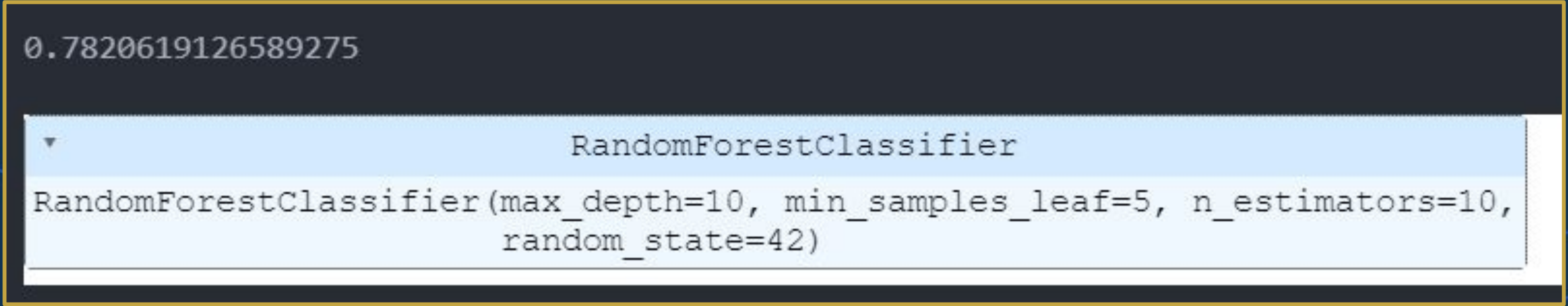
HyperionDe

Hyperparameter Tuning

Exhaustive search over specified parameter values for an estimator



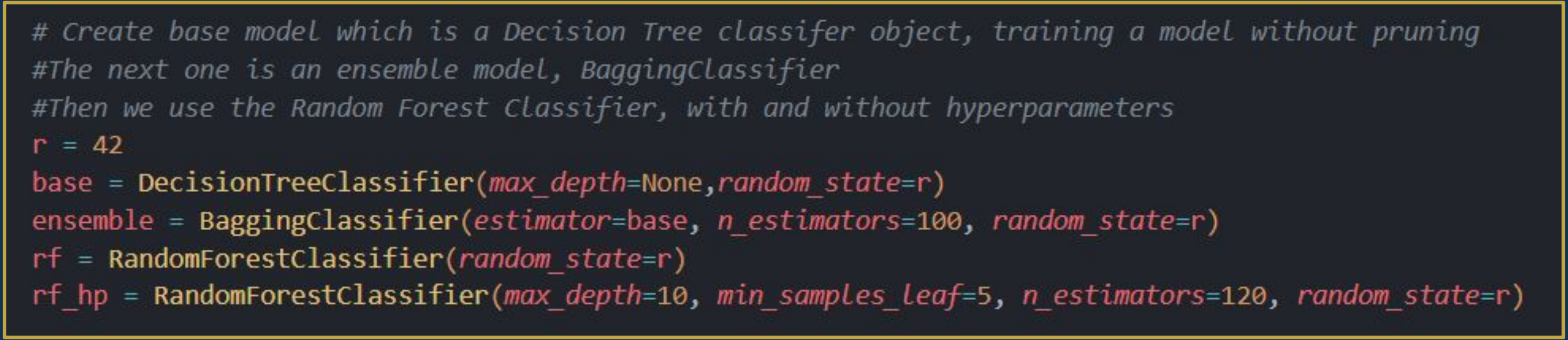
```
#Check best score hyperparameters
print(grid_search.best_score_)
rf_best = grid_search.best_estimator_
rf_best
```



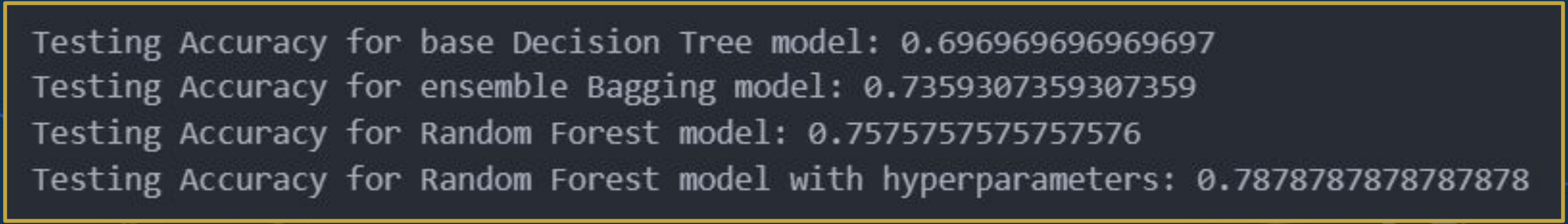
```
0.7820619126589275

▾ RandomForestClassifier
RandomForestClassifier(max_depth=10, min_samples_leaf=5, n_estimators=10,
                       random_state=42)
```

HyperionDev

Hyperparameter Tuning



```python
# Create base model which is a Decision Tree classifer object, training a model without pruning
#The next one is an ensemble model, BaggingClassifier
#Then we use the Random Forest Classifier, with and without hyperparameters
r = 42
base = DecisionTreeClassifier(max_depth=None,random_state=r)
ensemble = BaggingClassifier(estimator=base, n_estimators=100, random_state=r)
rf = RandomForestClassifier(random_state=r)
rf_hp = RandomForestClassifier(max_depth=10, min_samples_leaf=5, n_estimators=120, random_state=r)
```



```
Testing Accuracy for base Decision Tree model: 0.696969696969697
Testing Accuracy for ensemble Bagging model: 0.7359307359307359
Testing Accuracy for Random Forest model: 0.7575757575757576
Testing Accuracy for Random Forest model with hyperparameters: 0.7878787878787878
```

HyperionDev

Bias Variance Trade-Off

- **Balance between underfitting and overfitting.** Random Forest is better at managing bias-variance trade-off than Decision Tree.

- Decision Tree tends to have high variance, which can lead to overfitting, while Random Forest has a lower variance, which results in better generalisation.

- **max_features** and **min_samples_leaf** reduce correlation between trees, but might increase bias, since each tree now has less data to work with.

- Choose set of hyperparameters that nagivates this tradeoff between bias and variance to minimize error.

Summary

HyperionDev

Key Takeaways

1. Decision tree is more simple and interpretable but prone to overfitting, but a random forest is **complex** and **prevents** the risk of **overfitting**, better at **managing bias-variance trade-off**.
2. Random forest: more **robust** and **generalised** performance on new data, widely used in various domains, finance, healthcare, and deep learning.
3. **Features** that are ranked highly have a significant influence on the model's decision-making, improving its performance.
4. Random Forests are fast to train, but quite **slow** to create **predictions** once they are trained. In some real-world cases where **run-time performance** is important, other approaches would be preferred.
5. It is a **predictive** modeling tool, **not a descriptive tool.** For a description of the relationships in the data, other approaches would be better.

Further Resources

- https://www.geeksforgeeks.org/bagging-vs-boosting-in-machine-learning/
- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- Understanding Random Forests: From Theory to Practice https://arxiv.org/abs/1407.7502
- https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/

HyperionDev

Questions and Answers

HyperionDev

Thank you for attending

HyperionDev