Machine Learning

| Random Forest

Prof. Dr. Mostafa Elhosseini

https://youtube.com/ElhosseiniAcademy

Agenda

- Random Forest
- Spam and Ham emails Example
- Fitting a Random Forest manually
- Extra-Trees
- Feature Importance

Random Forest

- One of the **most well-known bagging** models.
- The **weak learners** are small **decision trees** trained on random subsets of the dataset.
- Random forests work well for classification and regression problems, and the process is similar

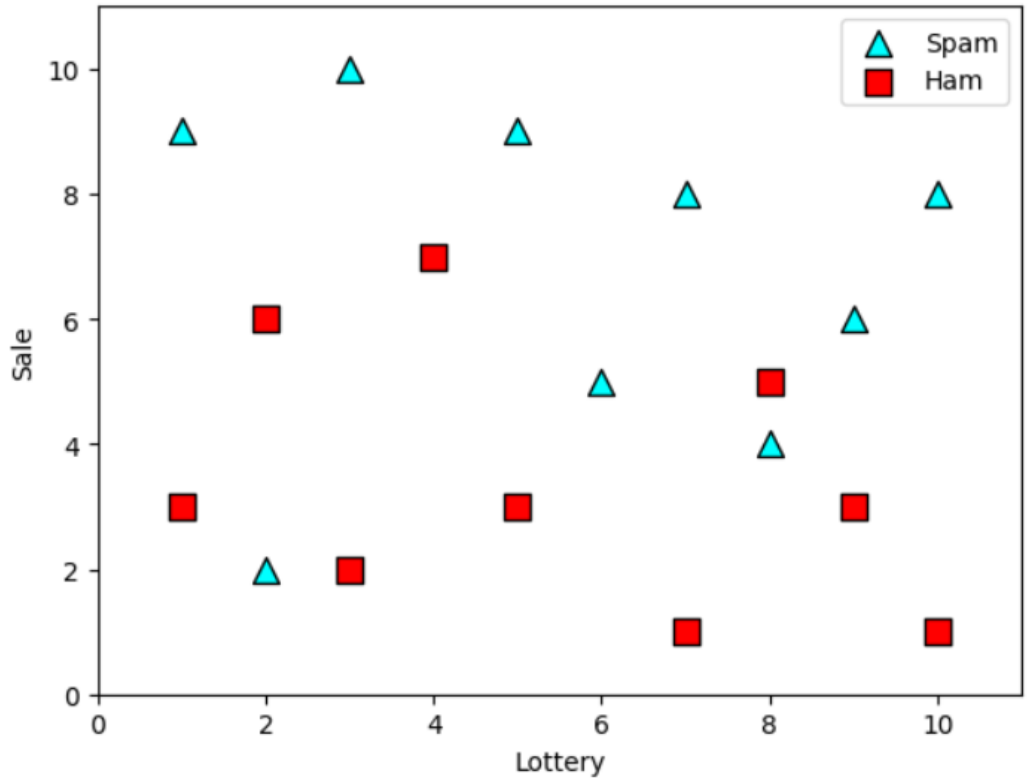
Example

- We use a small dataset of **spam and ham emails**
- The features of the dataset are the number of times the words “**lottery**” and “**sale**” appear in the email, and the “**yes/no**” label indicates whether the email is spam (yes) or ham (no).

Lottery	Sale	Spam
7	8	1
3	2	0
8	4	1
2	6	0
6	5	1
9	6	1
8	5	0
7	1	0
1	9	1
4	7	0
1	3	0
3	10	1
2	2	1
9	3	0
5	3	0
10	1	0
5	9	1
10	8	1

Example

```
features = spam_dataset[['Lottery', 'Sale']]
labels = spam_dataset['Spam']
utils.plot_points(features, labels)
```



Example

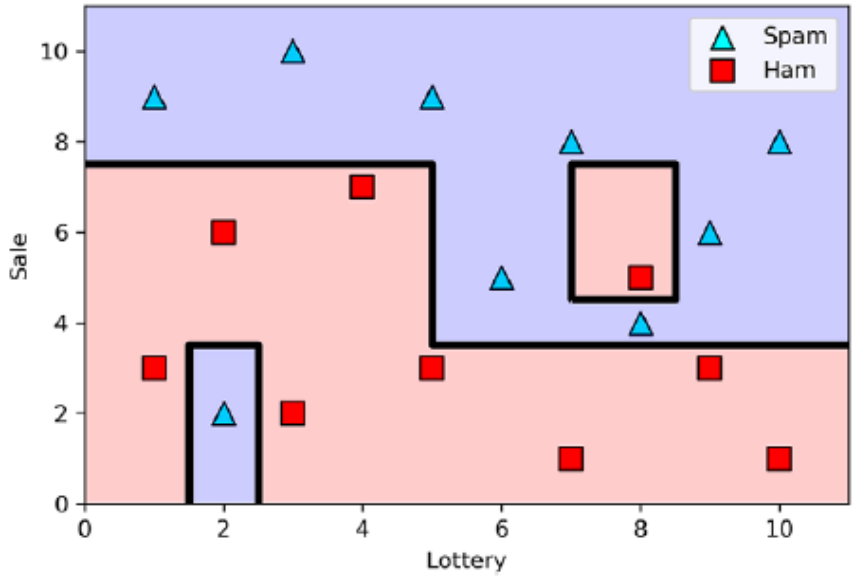
- let’s fit a **decision tree** classifier to this data and see how well it performs

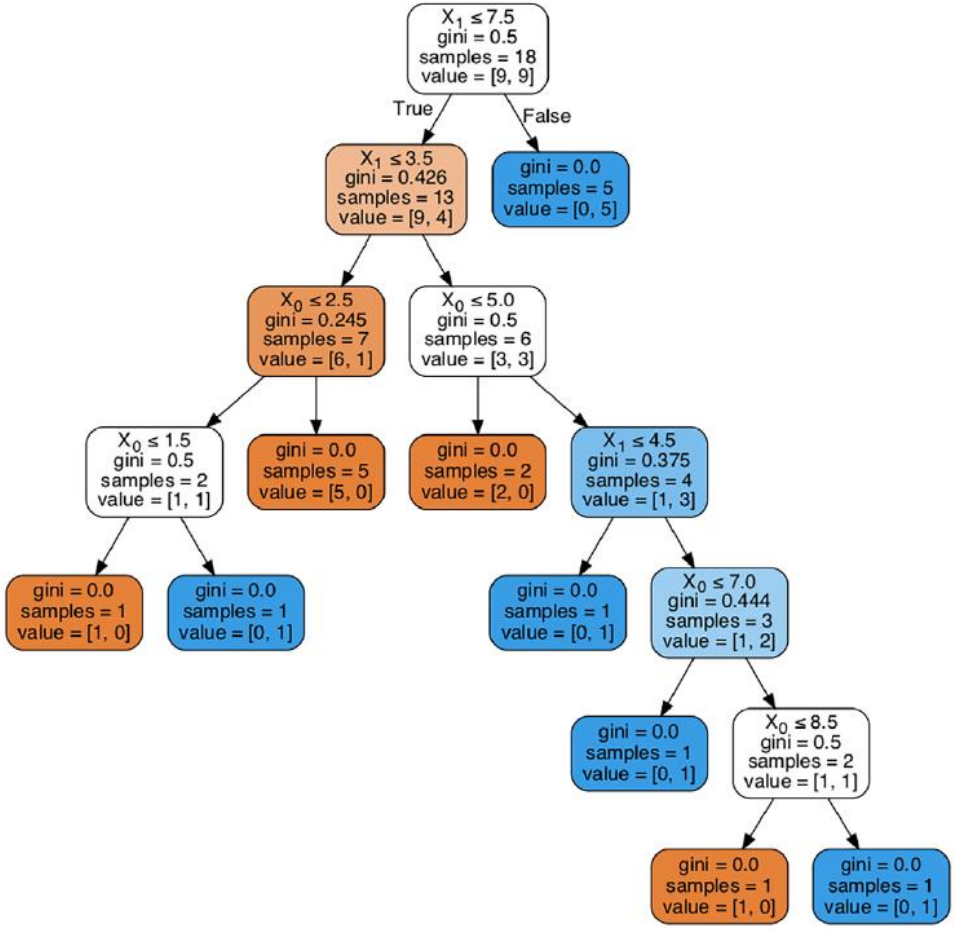
```
decision_tree_classifier = DecisionTreeClassifier(random_state=0)
decision_tree_classifier.fit(features, labels)
decision_tree_classifier.score(features, labels)
```

```
1.0
```

Example

- Notice that it fits the dataset very well, with a 100% training accuracy, although it **clearly overfits**





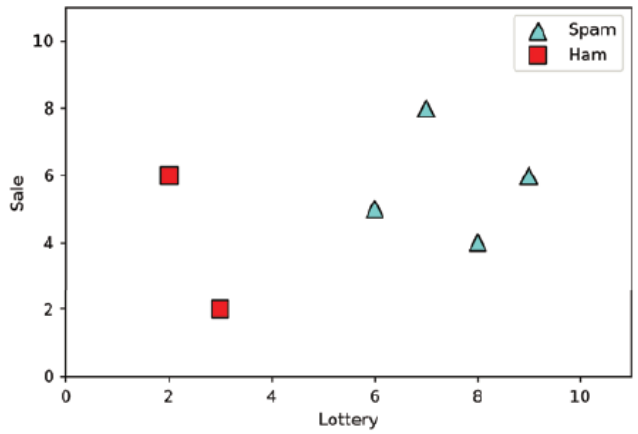
Fitting a Random Forest manually

- How to fit a random forest manually, although this is only for **educational purposes**, because this is not the way to do it in practice. In a nutshell, we **pick random subsets** from our dataset and **train a weak learner** (decision tree) on each one of them
- Some data points **may belong to several subsets**, and others **may belong to none**. The combination of them is our strong learner. The way the strong learner makes predictions is by letting the weak learners **vote**

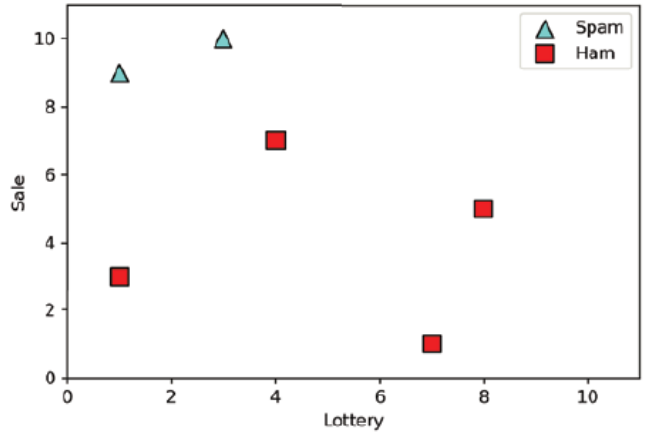
Fitting a random forest manually

- For this dataset, we use three weak learners. Because the dataset has 18 points, let's consider three subsets of 6 data points each

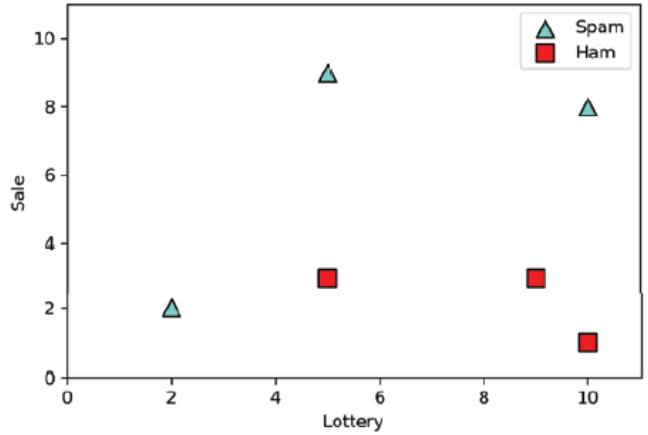
Subset 1



Subset 2

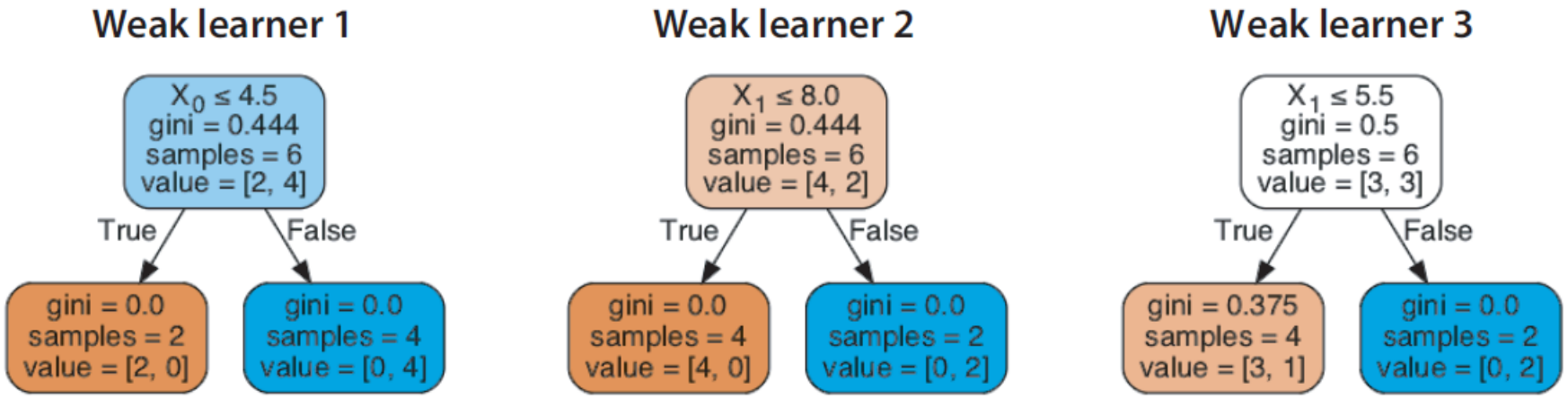


Subset 3



Fitting a random forest manually

- Fit a decision tree of depth 1 on each of these subsets
- Decision tree of depth 1 contains only one node and two leaves. Its boundary consists of a single horizontal or vertical line that splits the dataset as best as possible

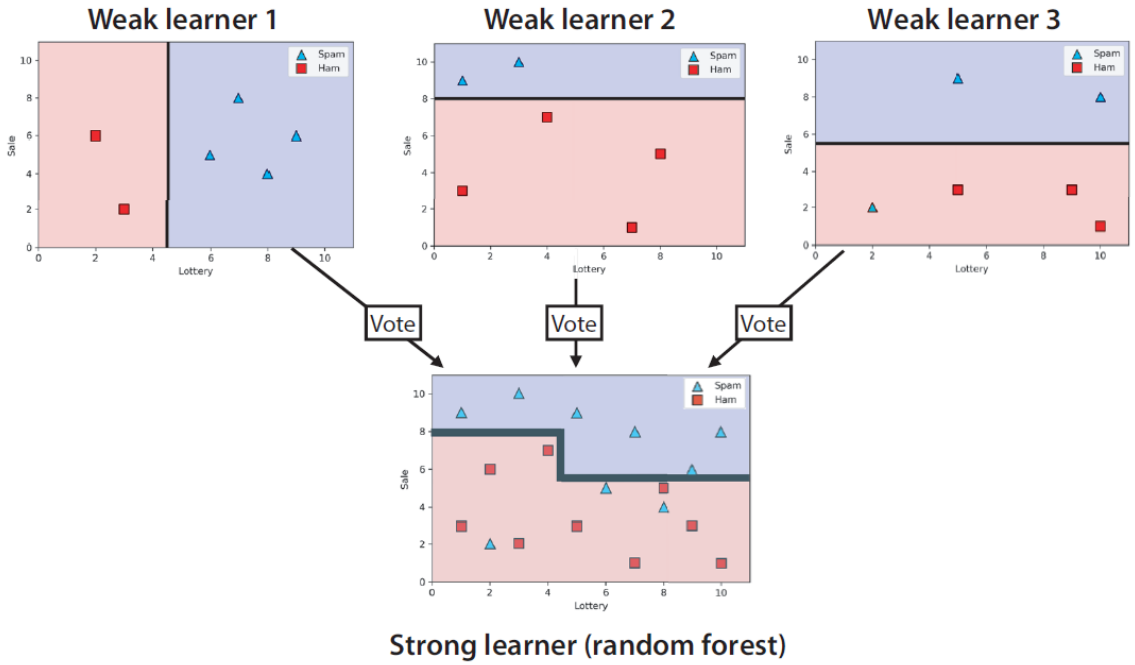


Fitting a random forest manually

- We combine these into a strong learner by voting. In other words, for any input, each of the weak learners predicts a value of 0 or 1. The prediction the strong learner makes is the most common output of the three

Fitting a random forest manually

- It allows a few mistakes in order to not overfit the data. However, we don't need to train these random forests manually, because Scikit-Learn has functions for this



Random Forests

- Random forest is an ensemble of decision trees, generally trained via the bagging method (or sometimes pasting), typically with max_samples set to the size of the training set.

- **RandomForestClassifier** class, which is more convenient and **optimized** for decision trees (similarly, there is a **RandomForestRegressor** class for regression tasks).

Random Forests

- Trains a random forest classifier with 500 trees, each limited to maximum 16 leaf nodes, using all available CPU cores

```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16,
                                 n_jobs=-1, random_state=42)
rnd_clf.fit(X_train, y_train)
y_pred_rf = rnd_clf.predict(X_test)
```

- With a few exceptions, a RandomForestClassifier has all the **hyperparameters** of a **DecisionTreeClassifier** (to control how trees are grown), plus all the hyperparameters of a **BaggingClassifier** to control the ensemble itself

Random Forests

- The random forest algorithm introduces extra randomness when growing trees; instead of searching for the very best feature when splitting a node, it searches for the **best feature among a random subset of features**.
- By default, it samples n features (where n is the total number of features). The algorithm results in greater tree diversity, which (again) trades a higher bias for a lower variance, generally yielding an overall better model

Training a random forest in Scikit-Learn

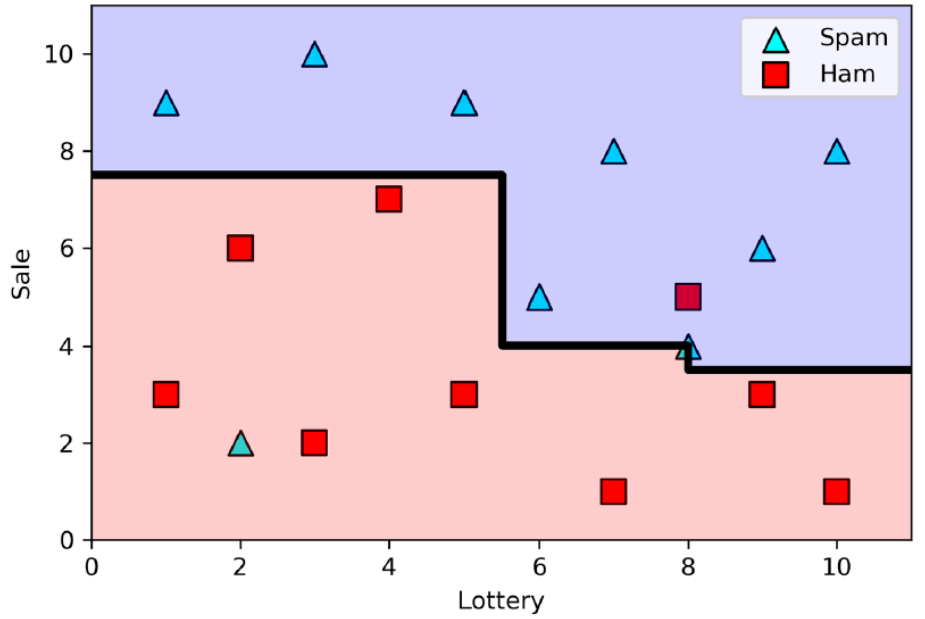
- Use of the RandomForestClassifier package

```
from sklearn.ensemble import RandomForestClassifier
random_forest_classifier = RandomForestClassifier(random_state=0, n_estimators=5, max_depth=1)
random_forest_classifier.fit(features, labels)
random_forest_classifier.score(features, labels)
```

- Five weak learners with the n_estimators hyperparameter. These weak learners are again decision trees, and we have specified that their depth is 1 with the max_depth hyperparameter

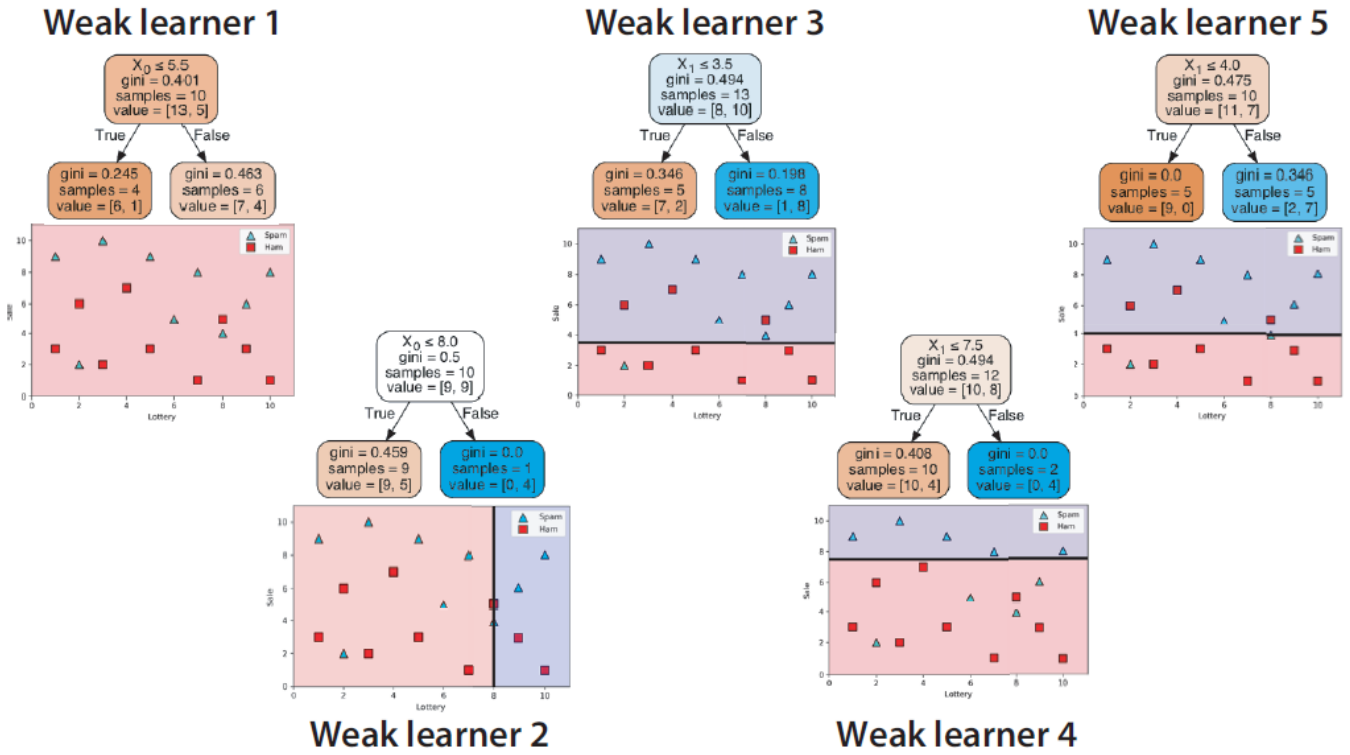
Training a random forest in Scikit-Learn

- Model makes some mistakes but manages to find a good boundary, where the spam emails are those with a lot of appearances of the words “lottery” and “sale” (top right of the plot) and the ham emails are those with not many appearances of these words (bottom left of the figure).



Training a random forest in Scikit-Learn

- Scikit-Learn also allows us to visualize and plot the individual weak learners



Training a random forest in Scikit-Learn

- Notice that **not all the weak learners are useful**. For instance, the first one classifies every point as ham

- In this section, we used **decision trees of depth 1** as weak learners, but in general, we can **use trees of any depth** we want.

- Try **retraining this model** using decision trees of higher depth by varying the max_depth hyperparameter, and see what the random forest looks like!

Random Forests

- The following **BaggingClassifier** is equivalent to the previous RandomForestClassifier

```
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(max_features="sqrt", max_leaf_nodes=16),
    n_estimators=500, n_jobs=-1, random_state=42)
```

Extra-Trees

- When you are growing a tree in a random forest, at each node only a random subset of the features is considered for splitting (as discussed earlier).

- It is possible to **make trees even more random** by also using **random thresholds for each feature** rather than searching for the best possible thresholds (like regular decision trees do).

- For this, simply set **splitter="random"** when creating a DecisionTreeClassifier.

Extra-Trees

- A forest of such extremely random trees is called an **extremely randomized trees** (or **extra-trees** for short) ensemble.

- This technique **trades more bias for a lower variance**. It also makes extra-trees classifiers **much faster to train** than regular random forests, because finding the best possible threshold for each feature at every node is one of the most time-consuming tasks of growing a tree.

Extra-Trees

- You can create an extra-trees classifier using Scikit-Learn's **ExtraTreesClassifier** class. Its API is identical to the RandomForestClassifier class, except **bootstrap defaults to False**.

- Similarly, the ExtraTreesRegressor class has the same API as the RandomForestRegressor class, except bootstrap defaults to False.

Extra-Trees

In Extra-Trees

- Instead of searching for the best possible split threshold for a chosen feature (as in Random Forests), thresholds are selected randomly.
- This further increases randomness and reduces computational cost.
- **Higher Bias**: By not choosing the optimal threshold, the individual trees are **less accurate**.
- **Lower Variance**: The additional randomness makes the ensemble **more robust**, as it **prevents overfitting** to the training data.
- **Extra-Trees take this a step further**, trading some accuracy (bias) for a significant reduction in variance, which can improve performance on noisy or high-dimensional datasets.

Feature Importance

- Random forests make it easy to measure the relative importance of each feature.

- Scikit-Learn measures a feature's importance by assessing how much tree nodes using that feature reduce impurity on average across all trees in the forest.

Feature Importance

- The importance is calculated as a weighted average, where each node's weight corresponds to the number of training samples associated with it.
- After training, Scikit-Learn automatically computes the importance score for each feature.
- The feature importance scores are scaled so that the total sum of all importances equals 1.

Feature Importance

- The most important features are the petal length (44%) and width (42%), while sepal length and width are rather unimportant in comparison (11% and 2%, respectively)

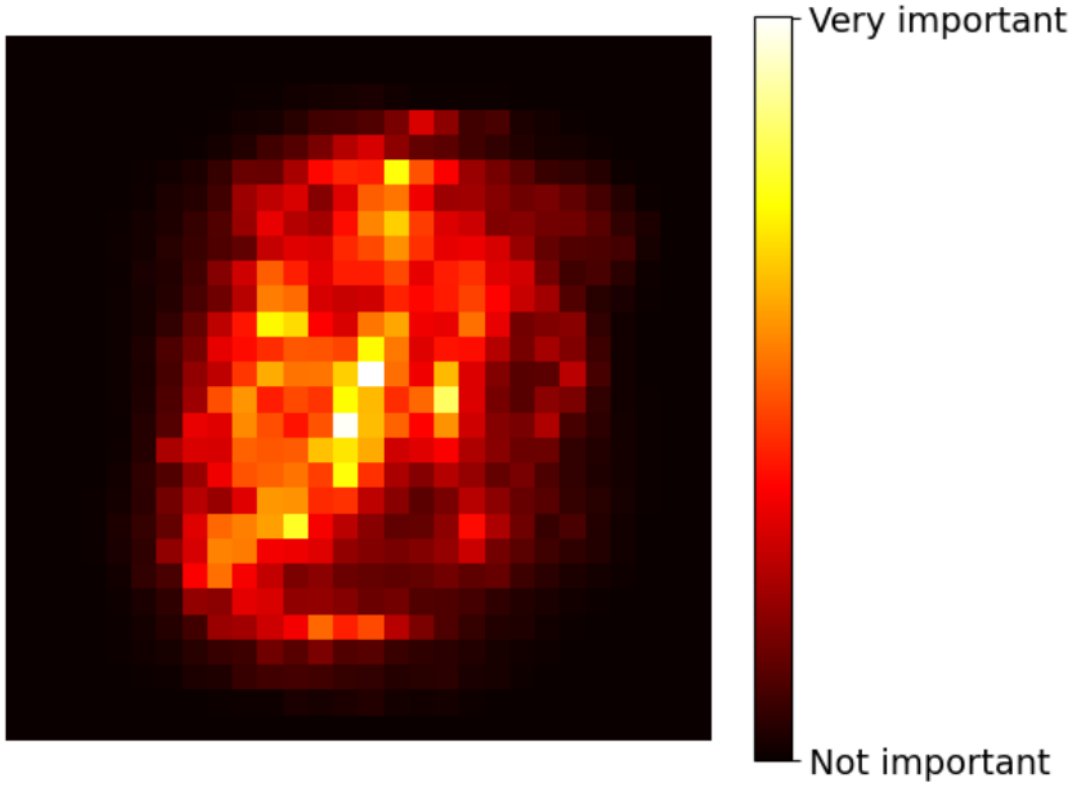
```
from sklearn.datasets import load_iris

iris = load_iris(as_frame=True)
rnd_clf = RandomForestClassifier(n_estimators=500, random_state=42)
rnd_clf.fit(iris.data, iris.target)
for score, name in zip(rnd_clf.feature_importances_, iris.data.columns):
    print(round(score, 2), name)
```

```
0.11 sepal length (cm)
0.02 sepal width (cm)
0.44 petal length (cm)
0.42 petal width (cm)
```

Feature Importance

- Train a random forest classifier on the MNIST dataset and plot each pixel's importance



Boosting

- we have to **take an exam** that consists of **100 true/false** questions on **many different topics**, including math, geography, science, history, and music. Luckily, we are allowed to call our **five friends**—Adriana, Bob, Carlos, Dana, and Emily—to help us. There is a small constraint, which is that all of them work **full time**, and they **don't have time to answer all 100 questions**, but they are more than happy to help us with a subset of them.

- What techniques can we use to get their help?

Next Lecture

- We give the exam to Adriana and ask her to answer only the questions she is the surest about.
- We assume that those answers are good and remove them from the test. Now we give the remaining questions to Bob, with the same instructions.
- We continue in this fashion until we pass it to all the five friends.