

Classical Data Analysis

Master in Big Data Solutions 2020-2021



Filipa Peleja

Víctor Pajuelo

filipa.peleja@bts.tech

victor.pajuelo@bts.tech

Today's class

Contents

1. Decision Trees recap
2. Ensemble Methods:
 1. Voting classifiers
 2. Bagging and pasting
 3. Bagging and out of bag evaluation
 4. Bagging and feature sampling
 5. Random Forests

Today's Objective

- Understand the power of ensemble methods
- Have an intuition of how random forests are built from decision trees
- Being able to use ensemble methods to improve single applied algorithms

Let's git things done!

Let's see it again

Pull Session 7 notebooks

```
$ git clone https://github.com/vfp1/bts-cda-2020.git
```

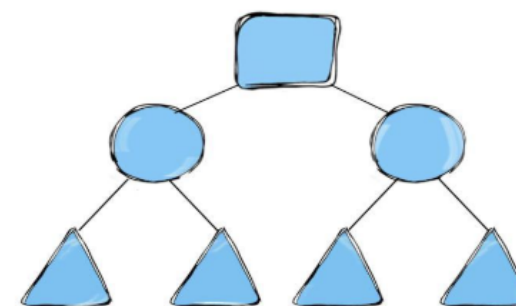
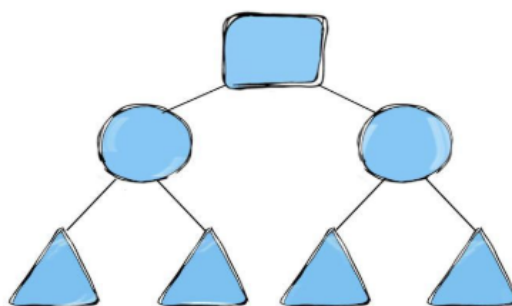
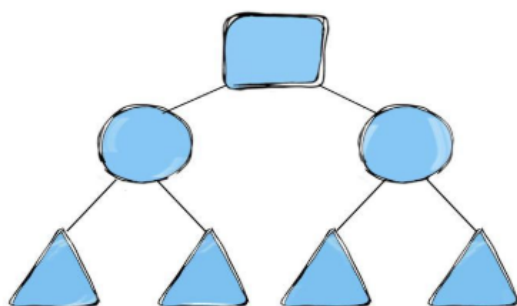
Decision Trees

Recap

Decision Trees

Recap

- Decision Trees (DTs from now on) are a powerful and versatile tool that can be used to solve a wide-range of problems
- They are really good for interpretability but they are really unstable
- They tend to overfit over a dataset since they “adapt” to it
- We explored the possibilities of building a “forest” of trees by averaging predictions over many trees

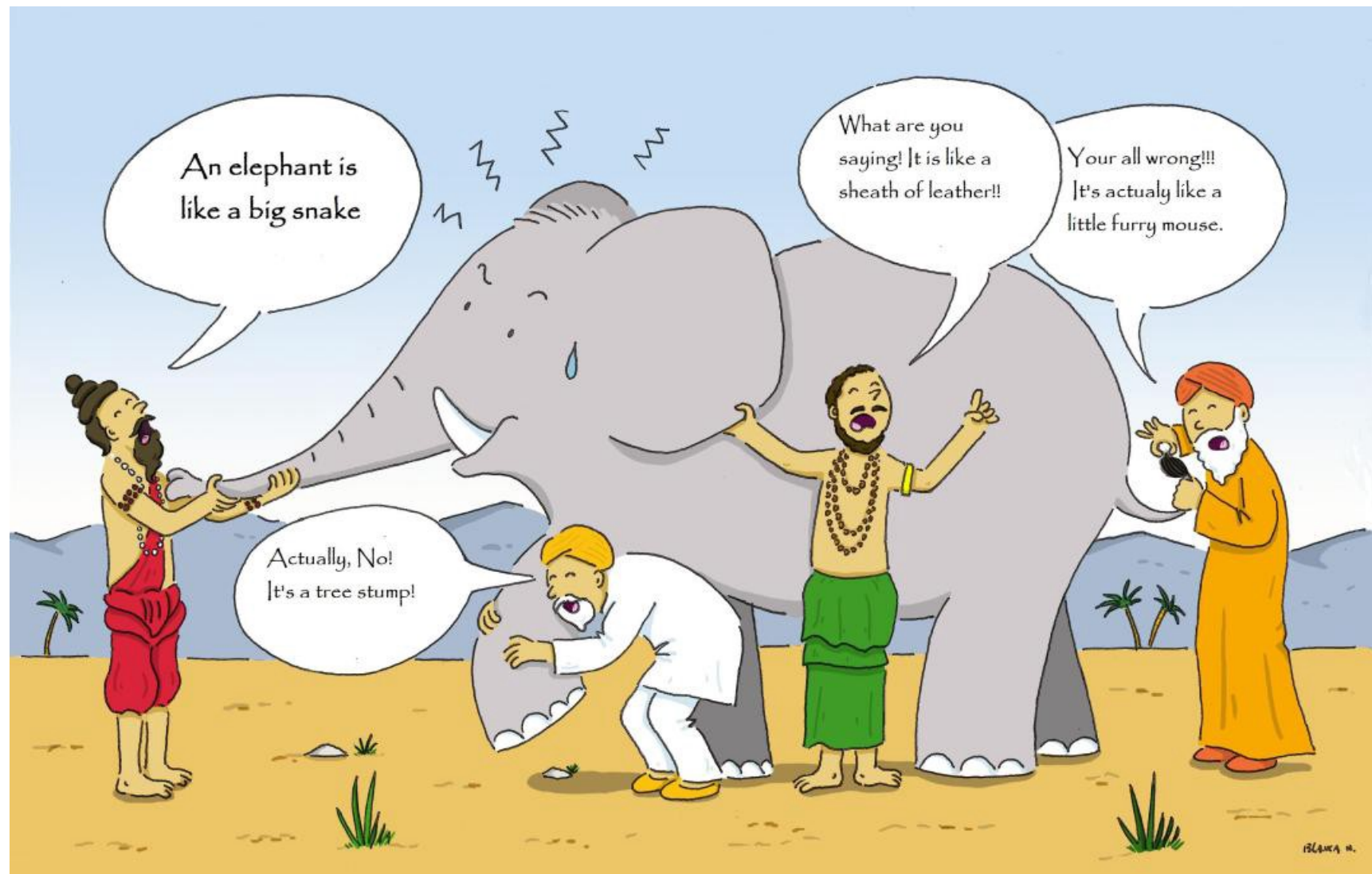


Decision Trees

Ensemble Learning

Ensemble Learning

An intuition



Ensemble Learning

Introduction

- If we **aggregate the predictions of a group predictors** (like classifiers or regressors), we will **usually get better predictions than with a single individual predictor**
- What is an **ENSEMBLE**? A group of predictors
- What is an **ENSEMBLE METHOD**? An Ensemble Learning algorithm

Ensemble Learning

When do we use ensemble methods?

- Usually, they are built towards the end of a project, it is kind of our “last bullet” to improve accuracy
 - We will see how sometimes Random Forests can be actually the “first bullet”
- Usually, they are built after we have tried several good predictors, and we combine them into a better predictor
- Ensemble methods are extremely powerful and are usually found at the top of ML competitions:
 - Kaggle
 - Netflix competition, etc.
- We will discuss the most popular Ensemble Methods, such as *bagging*, *boosting*, *stacking* and others, including Random Forests

Decision Trees

Voting classifiers

Ensemble Learning – Voting Classifiers

An intuition for ensemble methods

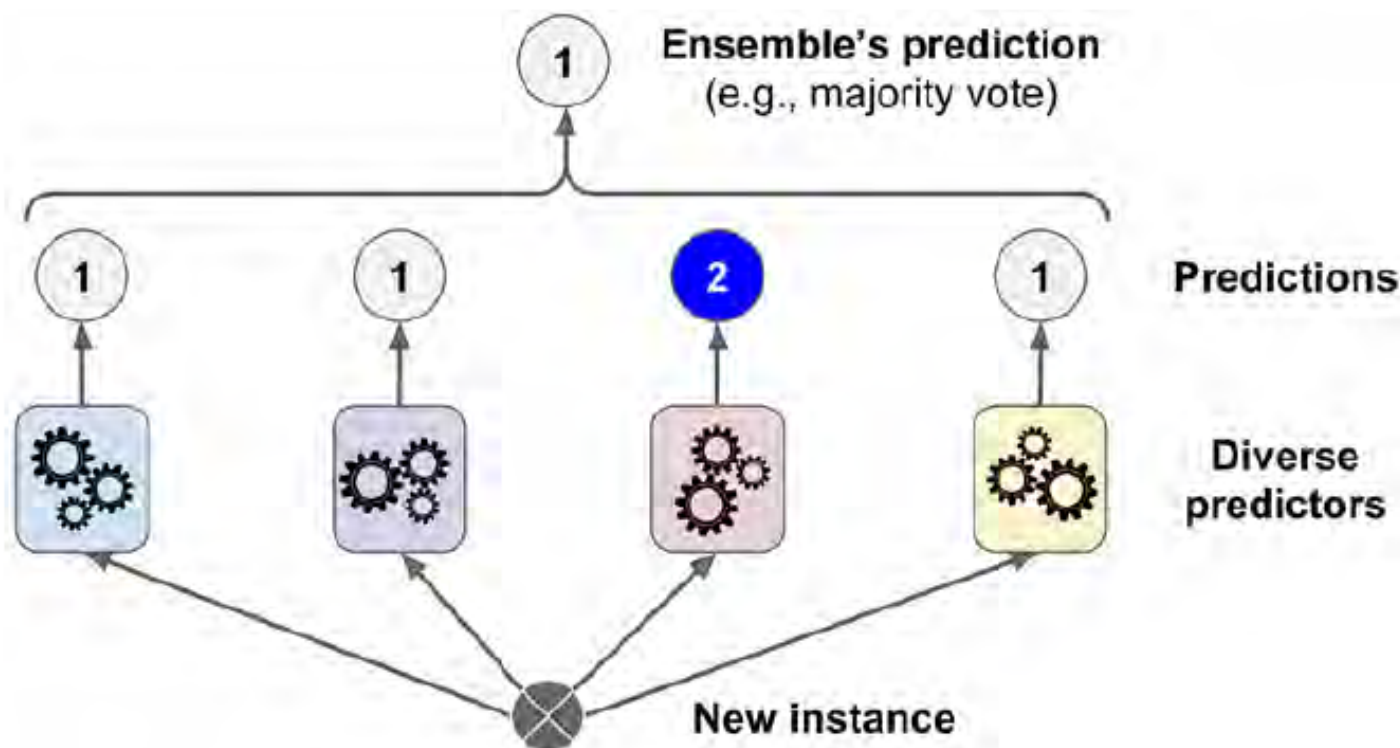
- Let's imagine that we have trained the following predictors:
 - A Logistic Regression classifier
 - A SVM classifier
 - A Random Forest classifier
 - A K-Nearest Neighbors classifier
 - Some Decision Trees
 - And perhaps more...

- We reach about 80% of accuracy at most with each one, how do we improve this?

Ensemble Learning – Voting Classifiers

An intuition for ensemble methods

- Following what we did in our exercise for last week, we can aggregate the predictions of each classifier and predict the class that gets the most votes
 - This **majority-voting** is called a ***hard voting classifier***



Ensemble Learning – Voting Classifiers

An intuition for ensemble methods

- As you probably saw while doing the exercise, the voting classifier usually **reaches higher accuracy** than the best classifier used in the ensemble, that means:
 - Even if each classifier is a weak learner (it just does slightly better than normal guessing), the ensemble can still be a strong learner (having high accuracy)



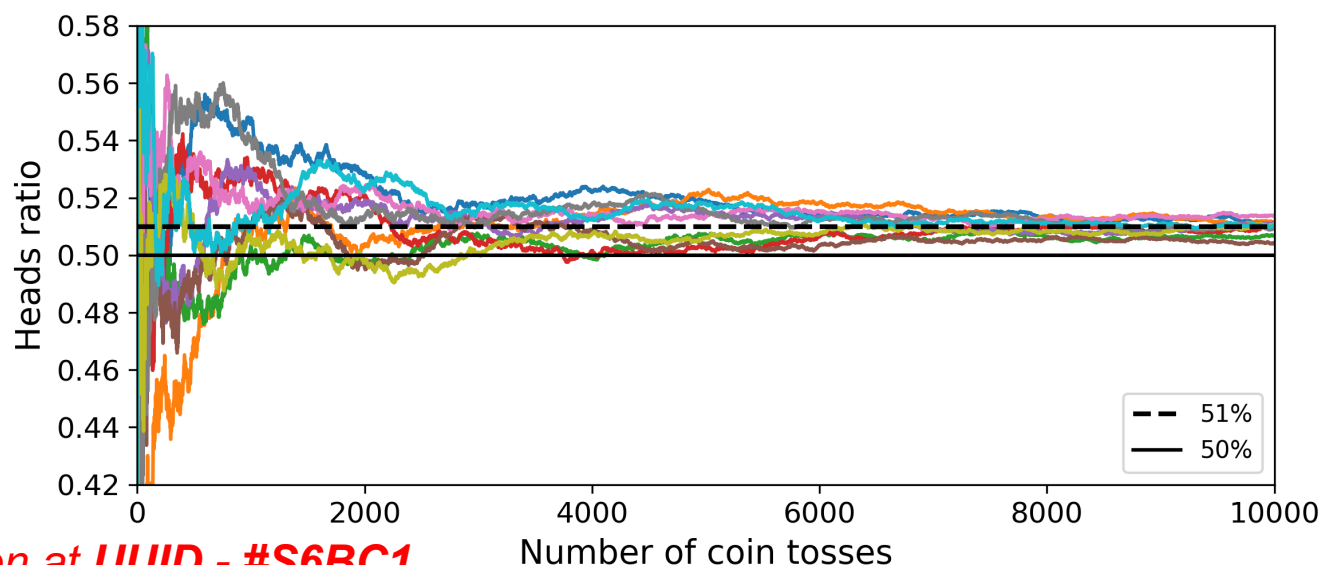
- However, this only occurs if there is a **SUFFICIENT NUMBER of WEAK LEARNERS** and they are **SUFFICIENTLY DIVERSE**

But, how is this actually possible?

Ensemble Learning – Voting Classifiers

Law of Large Numbers

- Imagine that we have a slightly biased coin that has a 51% chance of heads and 49% of tails
- After 1000 tosses, we will have about 510 heads and 490 tails
- The more we toss the coin, the higher probability we have to obtain a majority of heads overall

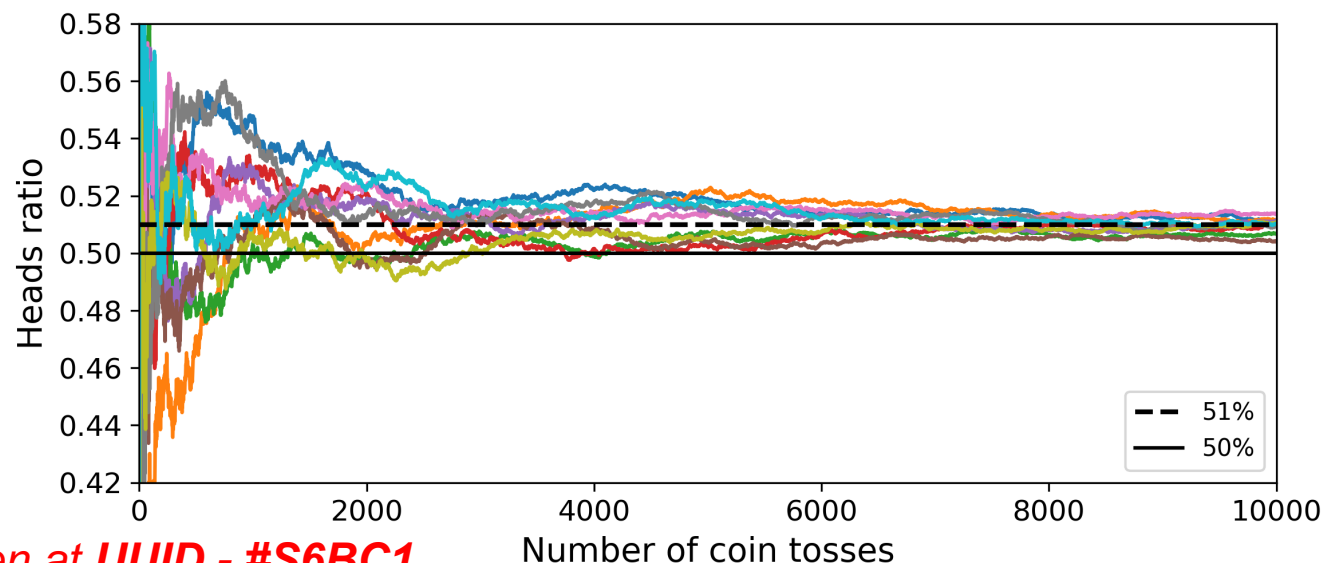


Code can be seen at **UUID - #S6BC1**

Ensemble Learning – Voting Classifiers

Law of Large Numbers

- This is called **the law of large numbers**, as you keep tossing the coin, the ratio of heads gets closer and closer to the probability of heads (51%)
- Eventually, all the 10 series below converge above tails at 51%



Code can be seen at **UUID - #S6BC1**

Ensemble Learning – Voting Classifiers

Law of Large Numbers for Ensemble Methods

- Now imagine that we build an ensemble containing 1000 classifiers that are individually correct only 51% of the time
- If we predict the majority voted class as in the 10 series for the coin toss, we get that we can improve our accuracy
- The ensemble's accuracy can go up to 75% accuracy at 1000 samples and beyond 97% at 10000 samples (due to the binomial distribution)
 - However, this only happens if all the classifiers are perfectly independent, making uncorrelated errors...
 - This usually is not the case, as they train over the same data
 - Individual predictors, are likely to repeat the same errors, so many majority votes will vote for the wrong class, reducing the ensemble's accuracy

Ensemble Learning – Voting Classifiers

Law of Large Numbers for Ensemble Methods



Ensemble methods perform the best when their predictors are as independent from each other as possible. One way to address this is to train an ensemble with **very different predictors**, so they will make **different type of errors**, improving the overall ensemble's accuracy

Ensemble Learning – Voting Classifiers

Hard Voting Classifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

log_clf = LogisticRegression(solver="lbfgs", random_state=42)
rnd_clf = RandomForestClassifier(n_estimators=100, random_state=42)
svm_clf = SVC(gamma="scale", random_state=42)

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='hard')
```

```
1 from sklearn.metrics import accuracy_score
2
3 for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
4     clf.fit(X_train, y_train)
5     y_pred = clf.predict(X_test)
6     print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
```

```
LogisticRegression 0.864
RandomForestClassifier 0.896
SVC 0.896
VotingClassifier 0.912
```

Code can be seen at **UUID - #S6BC2**

Ensemble Learning – Voting Classifiers

Hard Voting Classifier



A **hard voting** classifier takes the majority vote classification, that is, the majority class. But is susceptible then to take classes that are in low confidence. The **soft voting** classifier uses probabilities instead to give a weight of importance to each vote.

Ensemble Learning – Voting Classifiers

Soft Voting Classifier

```
log_clf = LogisticRegression(solver="lbfgs", random_state=42)
rnd_clf = RandomForestClassifier(n_estimators=100, random_state=42)
svm_clf = SVC(gamma="scale", probability=True, random_state=42)

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='soft')
voting_clf.fit(X_train, y_train)
```

```
1 from sklearn.metrics import accuracy_score
2
3 for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
4     clf.fit(X_train, y_train)
5     y_pred = clf.predict(X_test)
6     print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
```

```
LogisticRegression 0.864
RandomForestClassifier 0.896
SVC 0.896
VotingClassifier 0.92
```

But wait, SVM does not give probabilities... then how can we use soft voting with probabilities?

Ensemble Learning – Voting Classifiers

Soft Voting Classifier

- The **classifiers** that can estimate **probabilities** usually have a `predict_proba()` method
- If we can predict probabilities, is usually **best to use** `voting="soft"`, in order to predict the highest-class probability averaged over all the individual samples
 - Usually achieves higher performance than hard voting, since **it gives more weight to highly confident votes**
- **What about the SVM then?**
 - We need to set the `probability=True`
 - This makes the SVM use cross-validation to estimate the class probabilities, and will add a `predict_proba()` method
 - However, this will severely slow down training

Ensemble Learning

Bagging and Pasting

Ensemble Learning – Bagging and Pasting

Sampling Policy

- One way to improve a model is to use **very different training algorithms** independent from each other
- Another approach (the one we used in our exercise last week) is to use the **same training algorithm** for every predictor and train on **different random subsets** of the training set
- The selection or **sampling** of those subsets can either be done through bagging or pasting
- Defining the **sampling policy** is crucial for improving the accuracy in ensemble methods

Ensemble Learning – Bagging and Pasting

Sampling with/without replacement

- There are many policies for sampling, but the most common ones are to sample **with** and **without** replacement
 - Suppose we have a bowl of 100 unique numbers from 0 to 99. We want to select a random sample of numbers from the bowl. After we pick a number from the bowl, we can put the number aside or we can put it back into the bowl. If we put the number back in the bowl, it may be selected more than once; if we put it aside, it can be selected only one time.
- When a population element can be selected more than one time, we are **sampling with replacement**. When a population element can be selected only one time, we are **sampling without replacement**.

Ensemble Learning – Bagging and Pasting

Sampling with replacement

- Let's say that we have 7 numbers in a hat from 0 to 6 and we want to sample two:
 - If we use sampling with replacement (choosing one number and then putting it back to the hat) we can get something like...
 - $1,1 \rightarrow P(1,1) = 1/7 * 1/7 = .02$
 - $2,5 \rightarrow P(2,5) = 1/7 * 1/7 = .02$
 - $4,6 \rightarrow P(4,6) = 1/7 * 1/7 = .02$
 - $4,4 \rightarrow P(4,4) = 1/7 * 1/7 = .02$
 - Etc.
- Items are **independent**, one item does not affect the outcome of another. The probabilities of choosing one item are always the same.

Ensemble Learning – Bagging and Pasting

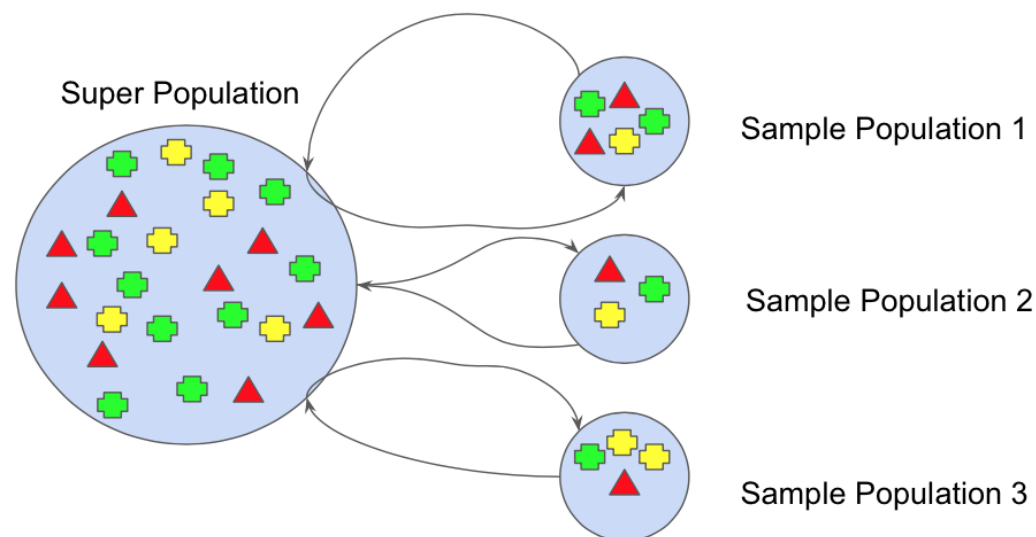
Sampling without replacement

- Let's say that we have 7 numbers in a hat from 0 to 6 and we want to sample two:
 - If we use sampling without replacement (choosing one number and keeping it) we can get something like
 - $1,2 \rightarrow P(1,2) = 1/7 * 1/6 = .024$
 - $1,3 \rightarrow P(1,3) = 1/7 * 1/6 = .024$
 - $4,5 \rightarrow P(4,5) = 1/7 * 1/6 = .024$
 - $4,0 \rightarrow P(4,0) = 1/7 * 1/6 = .024$
 - Etc.
- Items are **dependent**, choosing one item affects the possibilities of choosing another. However, it creates subsets that are always different, while *with replacement* the datasets can be repeated.

Ensemble Learning – Bagging and Pasting

Bagging

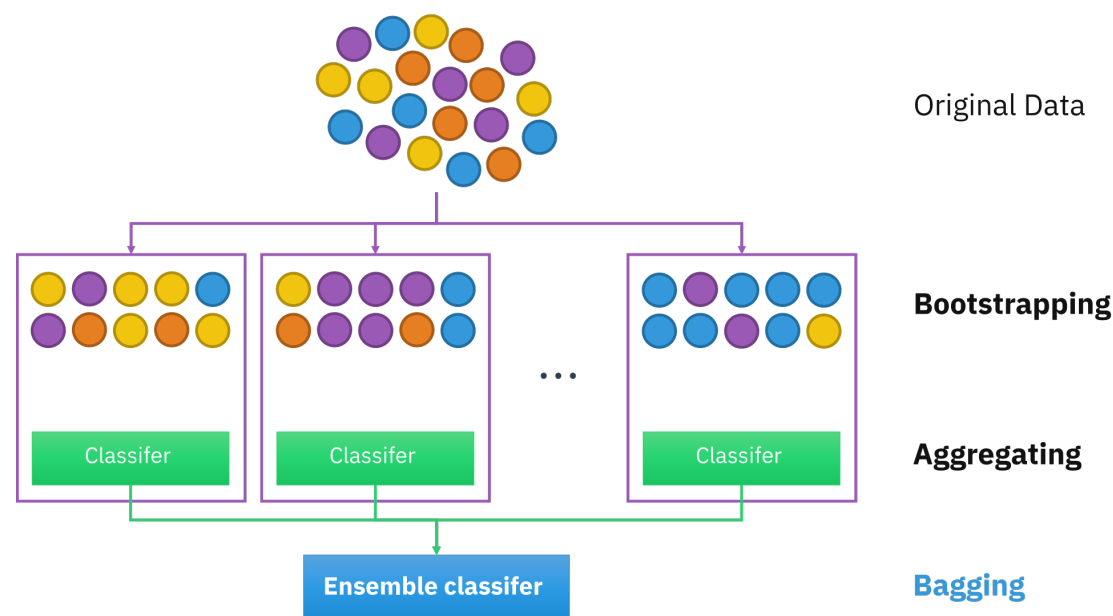
- Bagging stands for **bootstrap aggregating**
- In statistics, **bootstrap** is doing resampling with replacement
- When we use **bagging**, the sampling policy is performed ***with replacement***



Ensemble Learning – Bagging and Pasting

Bagging

- Bagging stands for **bootstrap aggregating**
- In statistics, **bootstrap** is doing resampling with replacement
- When we use **bagging**, the sampling policy is performed ***with replacement***



Breiman, L. Mach Learn (1996) 24: 123. <https://doi.org/10.1007/BF00058655>

Ensemble Learning – Bagging and Pasting

Pasting

- Pasting uses sampling **without replacement**
- Created by Leo Breiman, the same author that created bagging predictors
- Since each subset can be used once (while subsets of bagging can be used more than once), you need a **lot of data** for it to work
- Indeed, **pasting** was created for **addressing big databases**
 - The procedure takes small pieces of the data, grows a predictor on each small piece and then pastes these predictors together.
- **Bagging**, can have **repeated subsets**, which is great for smaller datasets, as it improves robustness

Breiman, L. Pasting Small Votes for Classification in Large Databases and On-Line. *Machine Learning* 36, 85–103 (1999) doi:10.1023/A:1007563306331

Ensemble Learning – Bagging and Pasting

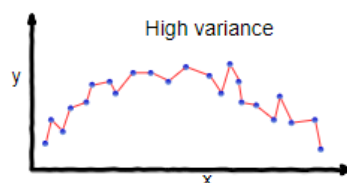
Aggregation

- In a nutshell, both bagging and pasting allow training samples to be sampled several times across multiple predictors
 - Only bagging allows training samples to be sampled several times for the same predictor
- When predictors are trained, the ensemble can make a prediction for a new instance just by **aggregating** the predictions of all predictors
- The **aggregation function** can be modified, but it is usually either:
 - Classification --> the *statistical mode* (the most frequent, like hard voting classifier)
 - Regression --> the average prediction

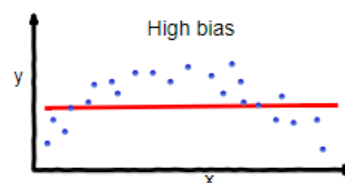
Ensemble Learning – Bagging and Pasting

Benefits

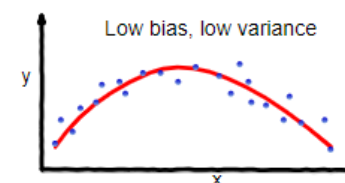
- Each individual predictor has higher bias than if trained with the whole training dataset
 - **Aggregation reduces both bias and variance**
- Usually, the result is that the ensemble tends to have **similar bias but lower variance** than a single predictor trained over the whole dataset
- Predictors can be trained and inferred in parallel over different CPU cores or servers
 - Bagging and pasting methods are **extremely scalable**



overfitting



underfitting



Good balance

Ensemble Learning – Bagging and Pasting

Implementation

- Both Bagging and Pasting are offered in Scikit-Learn through:
 - [BaggingClassifier](#) – for classification (obviously)
 - [BaggingRegressor](#) – for regression (obviously)
- Try to **identify what are we trying to do here:**

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(random_state=42), n_estimators=500,
    max_samples=100, bootstrap=True, random_state=42)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

Ensemble Learning – Bagging and Pasting

Implementation

- Yes! We are training:
 - A decision tree: `DecisionTreeClassifier()`
 - 500 Tree Classifiers: `n_estimators=500`
 - 100 training samples randomly sampled: `max_samples=100`
 - Sampled with replacement: an example of bagging (`bootstrap=True`)
 - If we wanted to use pasting, we needed (`bootstrap=False`)
 - Using all the CPU cores available: `n_jobs=-1`



Within `BaggingClassifier()` we can define `max_samples` as a float between 0 and 1. In this case, the numbers of instances to sample is the maximum number of instances in the dataset times the float number.

Ensemble Learning – Bagging and Pasting

Implementation

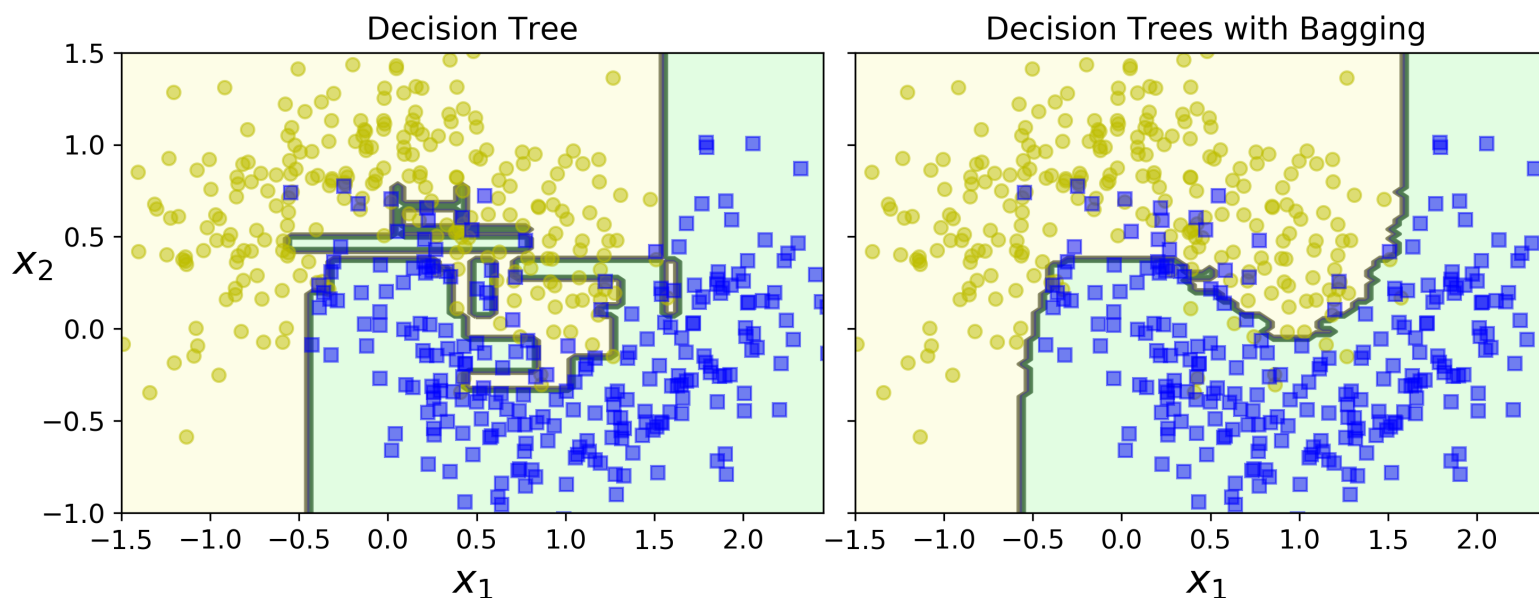


If the base algorithm can estimate class probabilities (i.e. has `predict_proba()` method), the `BaggingClassifier()` **automatically sets** soft voting instead of hard voting

Ensemble Learning – Bagging and Pasting

Effects on decision boundary

- The plot below represents a single tree classified and the bagging we just trained over the moons dataset
- The ensemble generalizes much better
 - Both have **comparable bias** but the ensemble has **way smaller variance**
 - They make the same number of errors, but the decision boundary at the ensemble is much more regular



Code can be seen at [UUID - #S6BC4](#)

Ensemble Learning – Bagging and Pasting

Bagging or Pasting

- We already know that pasting is usually used for really big datasets
 - Bagging also works for big datasets, by controlling the maximum number of samples
- Bootstrapping introduces more diversity in the subsets:
 - Results in **slightly higher bias than pasting**
 - Predictors are more independent, less correlated so **ensemble variance is reduced**
- Usually, **bagging usually results in better models**, so usually everyone goes for that



bootstrap=false

bootstrap=true

Ensemble Learning

Bagging and Out-of-Bag Evaluation

Bagging - OOB

Out-of-Bag Evaluation

- When we use bagging, some instances might be sampled many times for any predictor, while other instances may never be sampled.
- The remaining instances that are not sampled are called *out-of-bag* (OOB) instances
- Since an algorithm never sees OOB instances during training, they can be used for evaluation, without the need of separating a validation set or using Cross Validation.
- An ensemble can be evaluated by averaging out the OOB evaluations of each predictor

Bagging - OOB

Out-of-Bag Evaluation

- If we set `oob_score=True` when we create the `BaggingClassifier`, we have an automated evaluation after training
- To access the evaluation, call the `oob_score_` variable
- If you check that variable and the `accuracy_variable`, they should be about the same

```
1 bag_clf = BaggingClassifier(  
2     DecisionTreeClassifier(random_state=42), n_estimators=500,  
3     bootstrap=True, oob_score=True, random_state=40)  
4 bag_clf.fit(X_train, y_train)  
5 bag_clf.oob_score_  
  
0.9013333333333333
```

Code can be seen at **UUID - #S6BC5**

Ensemble Learning

Bagging and Feature Sampling

Bagging– Feature Sampling

Sampling Features!

- `BaggingClassifier()` also supports **sampling features**
- Controlled by the hyperparameters `max_features` and `bootstrap_features`.
 - This is the same as `max_samples` and `bootstrap`, but for feature sampling instead of instance sampling
- **SUPER USEFUL** for dealing with high dimensional data (images)
- Two main approaches for feature sampling...

Random Subspaces vs Random Patches

Random Subspaces

- Random Subspaces keeps all the instances but samples features
- Creates subsets of the training set that only contain certain features
- The chosen number of features are randomly sampled from the training set with replacement.

```
[[ 0.07377487 0.98297812 0.96381199 -0.14839061]
 [-1.60612919 1.26128902 0.16915662 0.60662095]
 [-0.24660218 0.77218249 1.36436104 1.23791409]
 [ 0.31787755 -0.97548711 0.44271172 -1.84485793]
 [ 1.27666678 0.36999326 -0.37374274 -1.02887293]
 [-0.81018955 -0.80938966 2.35680514 -0.48338745]
 [ 0.47108401 -0.29278008 -0.26851837 -2.23510265]
 [ 0.57911406 -0.24535111 -0.76438098 0.00808221]
 [ 0.69328831 -0.80603676 1.24371193 1.36884665]
 [ 1.17982921 0.27004813 0.36749512 0.72788058]
 [-0.05515918 0.5417384 -1.46265389 0.63611268]
 [-0.91725398 0.84667356 0.77556796 0.55255408]
 [ 1.24680939 -0.24020021 0.17075173 -0.34064414]
 [-0.48571915 0.66122176 0.05425974 0.12874584]
 [-2.26223332 0.80989996 0.11673109 -1.99900474]
 [ 0.60832379 -1.64353432 0.94054235 0.44946268]
 [-0.75636289 0.06237066 -0.44793654 -0.38747695]
 [-0.99186828 0.36427492 1.53901475 0.57631511]
 [ 0.88013213 0.13683464 -1.02933964 -0.73157095]
 [-1.71795755 -0.00791625 -0.64880648 1.67351386]
 [ 0.83745089 -0.33477737 -0.41567796 0.88031318]
 [-0.34017168 -1.06738938 -1.34639068 0.47034749]
 [-0.77993129 -0.17412887 -1.15965217 -0.62617248]
 [ 0.95723472 -0.64223762 1.57009179 -0.3161164 ]
 [-1.28484248 0.13544074 -1.81684153 -0.8524565 ]]
```


Random Patches

- Random Patches samples both the training Instances as well as the Features
- When the random subspace method is used along with bagging or pasting it is known as the random patches method.

```
[[ 0.07377487 0.98297812 0.96381199 -0.14839061]
 [-1.60612919 1.26128902 0.16915662 0.60662095]
 [-0.24660218 0.77218249 1.36436104 1.23791409]
 [ 0.31787755 -0.97548711 0.44271172 -1.84485793]
 [ 1.27666678 0.36999326 -0.37374274 -1.02887293]
 [-0.81018955 -0.80938966 2.35680514 -0.48338745]
 [ 0.47108401 -0.29278008 -0.26851837 -2.23510265]
 [ 0.57911406 -0.24535111 -0.76438098 0.00808221]
 [ 0.69328831 -0.80603676 1.24371193 1.36884665]
 [ 1.17982921 0.27004813 0.36749512 0.72788058]
 [-0.05515918 0.5417384 -1.46265389 0.63611268]
 [-0.91725398 0.84667356 0.77556796 0.55255408]
 [ 1.24680939 -0.24020021 0.17075173 -0.34064414]
 [-0.48571915 0.66122176 0.05425974 0.12874584]
 [-2.26223332 0.80989996 0.11673109 -1.99900474]
 [ 0.60832379 -1.64353432 0.94054235 0.44946268]
 [-0.75636289 0.06237066 -0.44793654 -0.38747695]
 [-0.99186828 0.36427492 1.53901475 0.57631511]
 [ 0.88013213 0.13683464 -1.02933964 -0.73157095]
 [-1.71795755 -0.00791625 -0.64880648 1.67351386]
 [ 0.83745089 -0.33477737 -0.41567796 0.88031318]
 [-0.34017168 -1.06738938 -1.34639068 0.47034749]
 [-0.77993129 -0.17412887 -1.15965217 -0.62617248]
 [ 0.95723472 -0.64223762 1.57009179 -0.3161164 ]
 [-1.28484248 0.13544074 -1.81684153 -0.8524565 ]]
```

Bagging– Feature Sampling

Approaches for Feature Sampling

 **!!** Feature sampling results in **higher predictor diversity**, trading a bit **more bias** for **lower variance**

Ensemble Learning

Voting,
Bagging/Pasting
recap

Ensemble Learning

Voting and Bagging/Pasting Recap

	Voting	Bagging/Pasting
Goal	To reach better accuracy through ensemble of different independent predictors (algorithms)	To reach better accuracy to ensemble of the same predictions over independent data subsets
Sklearn implementation	<u>VotingClassifier()</u>	<u>BaggingClassifier()</u>
Soft/Hard	Manually set	Automatically set
OOB?	Nope, only algorithms are “sampled”	Yep, instances are sampled
Feature sampling	Nope, only algorithms are combined here!	Yes, small subsets of data are combined here, so we can subset features as well

Ensemble Learning

Random Forests

Random Forests

Introduction

- A Random Forest is nothing but an **ensemble** of decision trees.

*Decision trees are attractive classifiers due to their high execution speed. But trees derived with traditional methods often cannot be grown to arbitrary complexity for possible **loss of generalization accuracy on unseen data**. The limitation on complexity usually means suboptimal accuracy on training data. Following the principles of stochastic modeling, we propose a method to construct tree-based classifiers whose capacity can be arbitrarily expanded for increases in accuracy for both training and unseen data. The essence of the method is to **build multiple trees in randomly selected subspaces of the feature space**. Trees in, different subspaces generalize their classification in complementary ways, and their combined classification can be monotonically improved. The validity of the method is demonstrated through experiments on the recognition of **handwritten digits**.*



[“Random Decision Forests”, T.Ho \(1995\)](#)

Random Forests

Introduction

- A Random Forest is nothing but an **ensemble** of decision trees.
- Usually trained through **bagging method** (sometimes pasting as well) and with **max_samples** to the size of the training dataset.
- Indeed, we can use a `BaggingClassifier` with a `DecisionTreeClassifier` in order to build a Random Forest. However, the `RandomForestClassifier` class is optimized for Decision Trees.
- There is also the `RandomForestRegressor` class

Random Forests

Introduction

- The `RandomForestClassifier` hyperparameters are derived from:
 - `DecisionTreeClassifier`: to control how trees are grown
 - `BaggingClassifier`: to control the ensemble itself
- With some exceptions...
 - `splitter` : disappeared, forced to “`random`”
 - `presort` : disappeared, forced to “`False`”
 - `max_samples` : disappeared, forced to `1.0`
 - `base_estimator` : disappeared, forced to `DecisionTreeClassifier`

Random Forests

Introduction

```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, random_state=42)
rnd_clf.fit(X_train, y_train)

y_pred_rf = rnd_clf.predict(X_test)
```

VS

```
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(splitter="random", max_leaf_nodes=16, random_state=42),
    n_estimators=500, max_samples=1.0, bootstrap=True, random_state=42)
```

Random Forests

Introduction

- Random Forest introduces actually **even more randomness** while growing trees
 - Instead of searching the best feature for splitting a node, random forests search the best feature among a random subset of features
- Random Forests have greater tree diversity
 - Trades for higher bias for lower variances
 - But it has a more generalistic and overall better model

Random Forests

Extra Trees Classifier

- Tree growing in a Random Forest, at each node only a random subset of the features is considered for splitting
- There is a possibility to further randomize trees by using **random thresholds** for each feature, rather than optimizing for the best threshold
 - This is the Extremely Randomized Trees ensemble
 - The algorithm is available through the ExtraTreesClassifier class
- This algorithm also trades adding more bias for a lower variance, and also **trains faster**, since finding the best possible threshold for each feature at every node is the most consuming task while growing a tree

Which one is better? The best is to run both through GridSearchCV and compare results

Summary

Decision Trees

VS

Random Forests

VS

Extra Trees

Decision Tree vs Random Forest

Summary

- **Decision Tree (High Variance)**
 - A single decision tree is usually overfits the data it is learning from because it learn from only one pathway of decisions. Predictions from a single decision tree usually don't make accurate predictions on new data.
- **Random Forest (Medium Variance)**
 - Random forest models reduce the risk of overfitting by introducing randomness by:
 - building multiple trees (n_estimators)
 - drawing observations with replacement (i.e., a bootstrapped sample)
 - splitting nodes on the best split among a **random subset** of the features selected at every node

Extra Trees (Low Variance)

- Extra Trees is like Random Forest, in that it builds multiple trees and splits nodes using random subsets of features, but with two key differences:
 - builds multiple trees with **bootstrap = False** by default, which means it samples without replacement
 - nodes are split based on **random** splits among a **random subset** of the features selected at every node
- In Extra Trees, randomness doesn't come from bootstrapping of data, but rather comes from the random splits of all observations.
- ExtraTrees is named for (Extremely Randomized Trees).

Feature Importance

Random Forests

Feature Importance

- If we look at a single Decision Tree, **important features** are likely to appear closer to the **root of the tree**, while unimportant features will often appear closer to the leaves (or not at all).
- It is therefore possible to get an **estimate of a feature's importance** by computing the **average depth at which it appears** across all trees in the forest.
- Scikit-Learn computes this automatically for every feature after training, we can access it using the **feature_importances_** variable.

Random Forests

Feature Importance

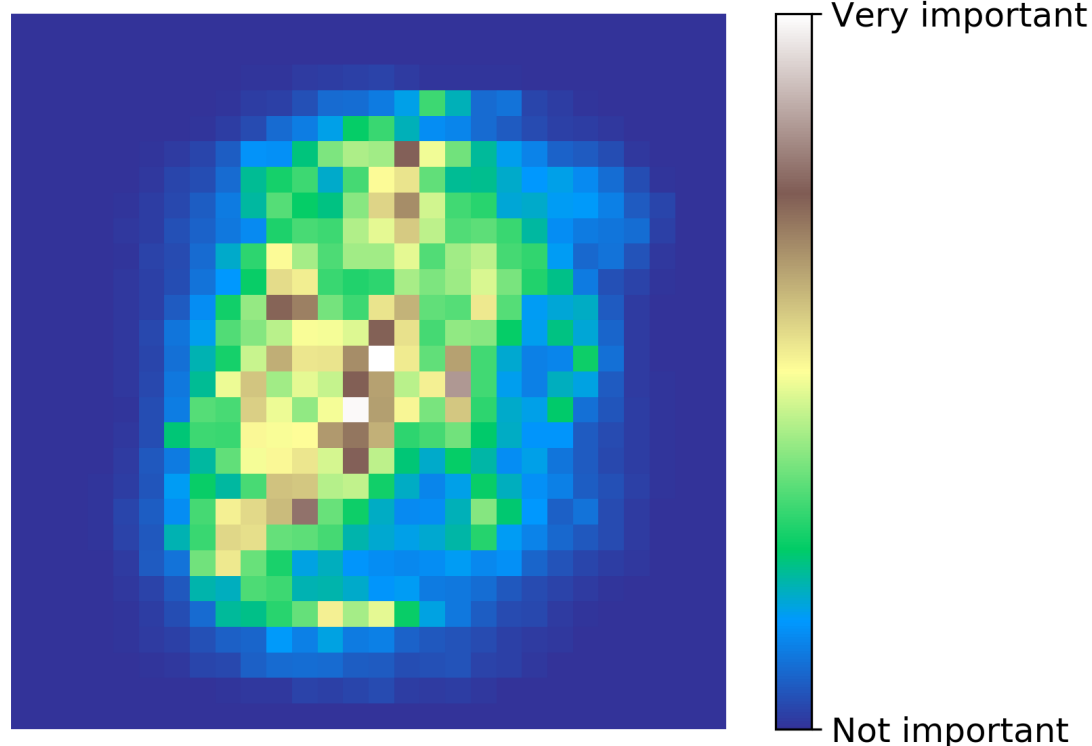
```
1 from sklearn.datasets import load_iris
2 iris = load_iris()
3 rnd_clf = RandomForestClassifier(n_estimators=500, random_state=42)
4 rnd_clf.fit(iris["data"], iris["target"])
5 for name, score in zip(iris["feature_names"], rnd_clf.feature_importances_):
6     print(name, score)
```

```
sepal length (cm) 0.11249225099876375
sepal width (cm) 0.02311928828251033
petal length (cm) 0.4410304643639577
petal width (cm) 0.4233579963547682
```

Random Forests

Most Important Features in MNIST

```
image = rnd_clf.feature_importances_.reshape(28, 28)
plt.imshow(image, cmap = mpl.cm.terrain, interpolation="nearest")
plt.axis("off")
cbar = plt.colorbar(ticks=[rnd_clf.feature_importances_.min(), rnd_clf.feature_importances_.max()])
cbar.ax.set_yticklabels(['Not important', 'Very important'])
plt.show()
```



Code can be seen at **UUID - #S6BC7**

Random Forests

Feature Importance



Random Forests are actually **extremely useful at showing which feature matters**, just in case we need to **perform feature selection**

Random Forests

Summary and Hyperparameters

Random Forests

Tips on usage

- Most of the hyperparameters are shared with Decision Trees
- Samples are drawn with replacement if `bootstrap=True`
- It is recommended to use OOB to check the generalization accuracy of the forest

Ensemble Learning

In class exercises

Ensemble learning

In class exercises

Go to the notebook

RECAP

Resources

Important resources

- Lex Friedman Series (MIT)
- Sklearn docs
- Aurelien Geron, “Hands-on machine learning with scikit-learn, Keras & Tensorflow”
- “Ensembles on Random Patches,” G. Louppe and P. Geurts (2012).
- “The random subspace method for constructing decision forests,” Tin Kam Ho (1998).
- “Random Decision Forests,” T. Ho (1995).
- “Extremely randomized trees,” P. Geurts, D. Ernst, L. Wehenkel (2005).
- Breiman, L. Mach Learn (1996) 24: 123. <https://doi.org/10.1007/BF00058655>
- Breiman, L. Pasting Small Votes for Classification in Large Databases and On-Line. *Machine Learning* 36, 85–103 (1999) doi:10.1023/A:1007563306331

