Méthodes d'ensemble

TP 1 – Random Forest

LE Do Thanh Dat, YOU Borachhun

Exercice 1 : Random Forest

1. Importer les libraries

[43]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
```

2. Importer les données du Titanic, et filtrer les observations avec NA

[44]:
```
df = pd.read_csv('./data/titanic.csv')

string_list = [each_string.lower() for each_string in df.columns]
df.columns = string_list
df.dropna(inplace=True)

df.head(5)
```

[44]:
```
    passengerid  survived  pclass  \
1             2         1       1
3             4         1       1
6             7         0       1
10           11         1       3
11           12         1       1

                                                 name     sex   age  sibsp  \
1   Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
3        Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
6                             McCarthy, Mr. Timothy J    male  54.0      0
10                    Sandstrom, Miss. Marguerite Rut  female   4.0      1
11                           Bonnell, Miss. Elizabeth  female  58.0      0

    parch    ticket     fare cabin embarked
1       0  PC 17599  71.2833   C85        C
3       0    113803  53.1000  C123        S
```

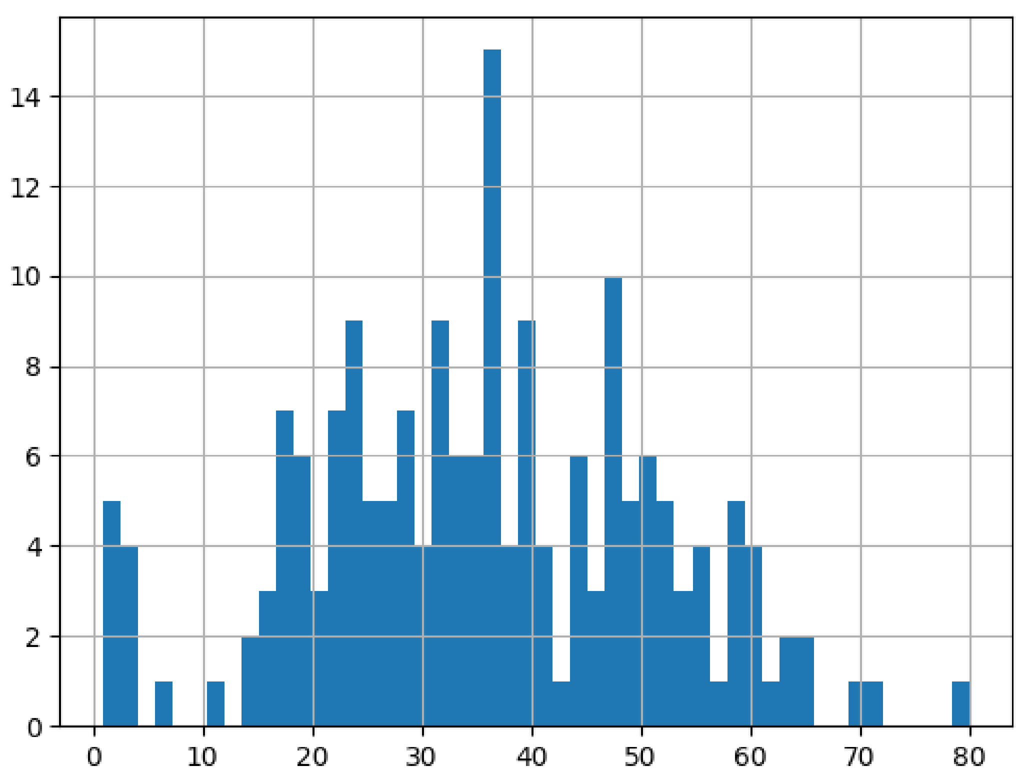
```
6        0     17463   51.8625    E46         S
10       1   PP 9549   16.7000     G6         S
11       0    113783   26.5500   C103         S
```

3. Prenez connaissance des quelques features dans le dataframe

[45]:
```python
print('----- Classes -----')
print('Classes: ', df['pclass'].unique())
print('Passengers by class:\n', df['pclass'].value_counts().values)
print('----- Genre -----')
print('Genre: ', df['sex'].unique())
print('Passengers by sex:\n', df['sex'].value_counts().values)

df['age'].hist(bins=50)
```

```
----- Classes -----
Classes:  [1 3 2]
Passengers by class:
 [158  15  10]
----- Genre -----
Genre:  ['female' 'male']
Passengers by sex:
 [95 88]
```

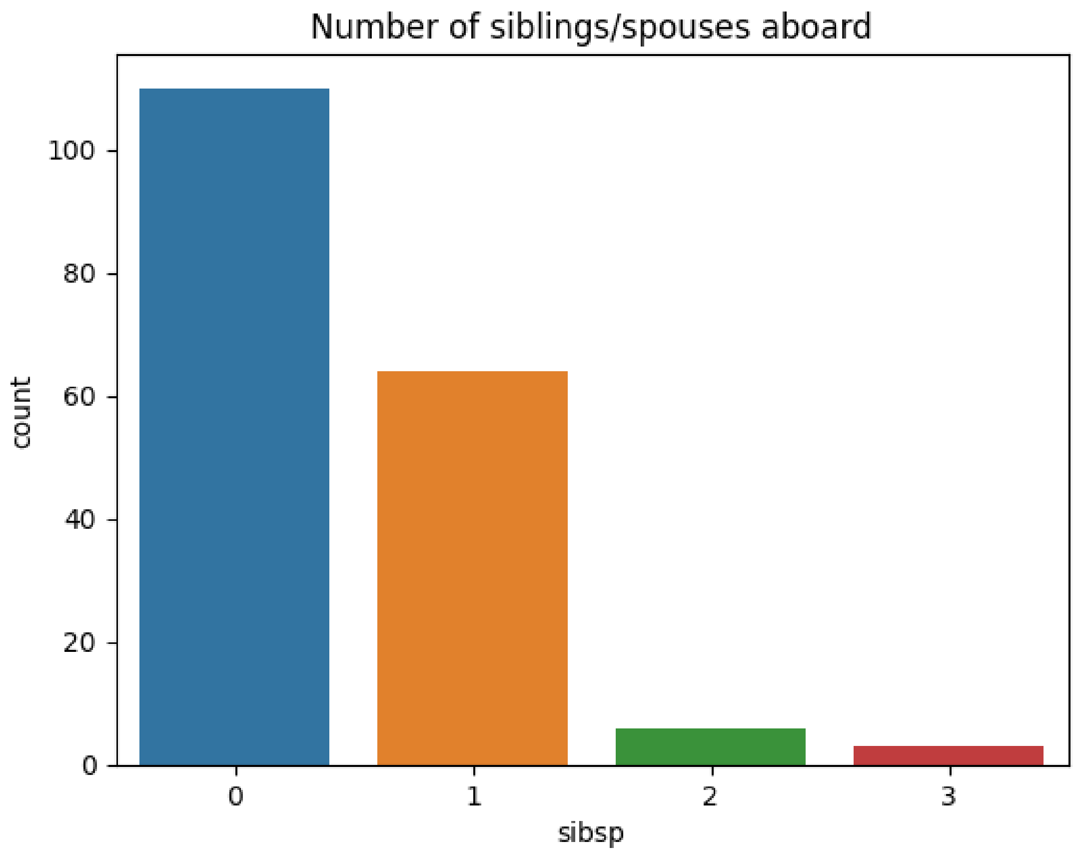
[45]: <Axes: >



Autre visualisation que l'on peut faire :

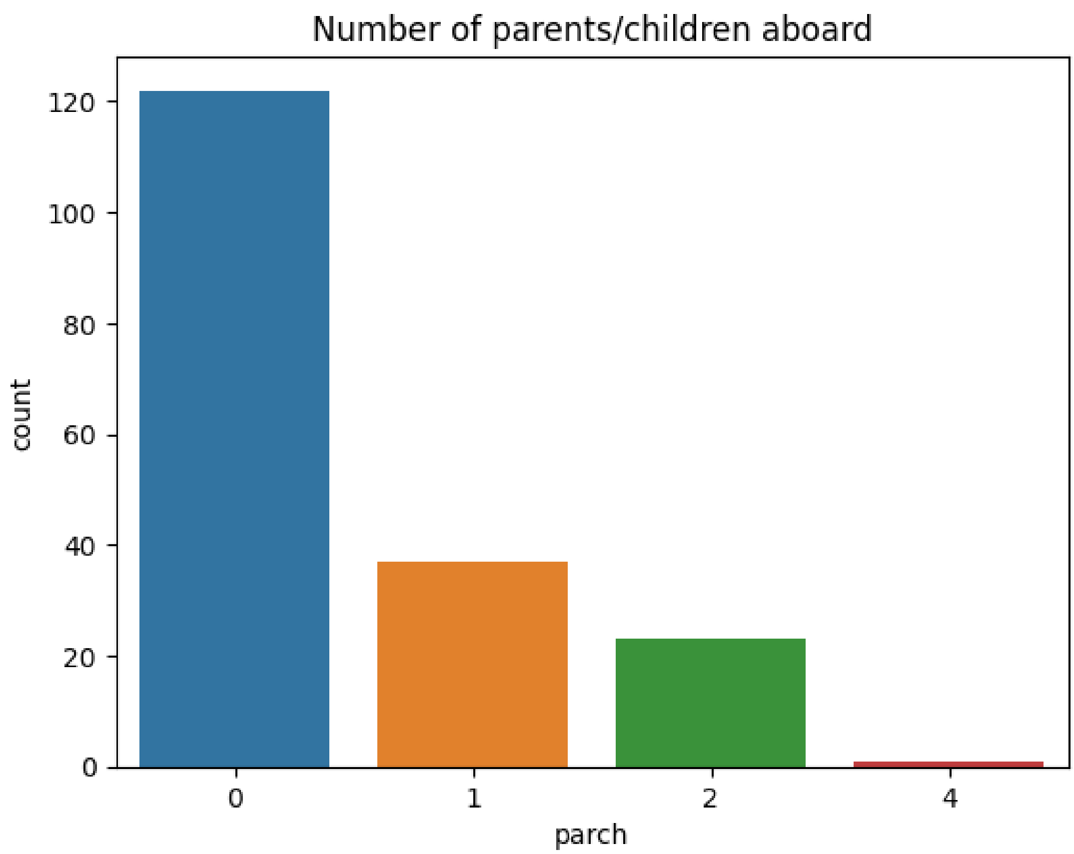
[46]:
```
sns.countplot(df, x='sibsp').set(title='Number of siblings/spouses aboard')
```

[46]: [Text(0.5, 1.0, 'Number of siblings/spouses aboard')]



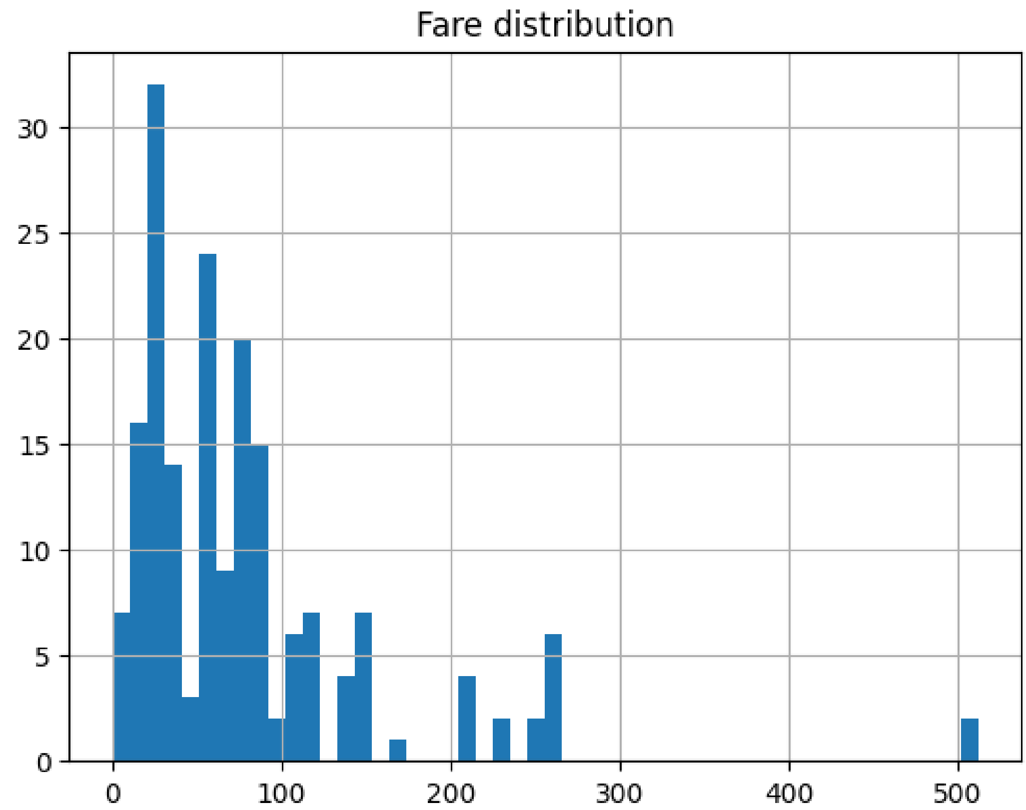
[47]:
```
sns.countplot(df, x='parch').set(title='Number of parents/children aboard')
```

[47]: [Text(0.5, 1.0, 'Number of parents/children aboard')]



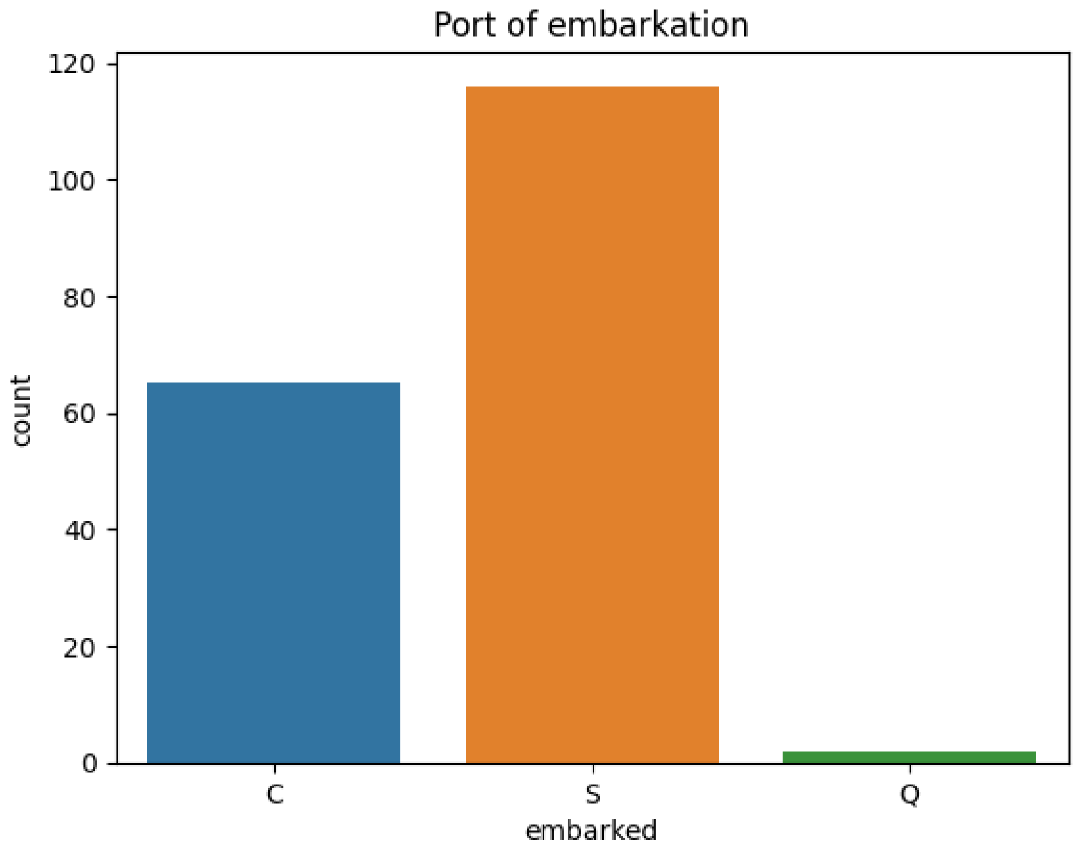
[48]:
```
df['fare'].hist(bins=50)
plt.title('Fare distribution')
```

[48]: Text(0.5, 1.0, 'Fare distribution')



[49]:
```
sns.countplot(df, x='embarked').set(title='Port of embarkation')
```

[49]: [Text(0.5, 1.0, 'Port of embarkation')]



4. Preprocessing des données : binarisations des quelques features catégoriels

[50]:
```
# Binarize columns
X = df[['pclass', 'sex', 'age', 'embarked']].copy()

# Using Sklearn to binarize
from sklearn import preprocessing
lb = preprocessing.LabelBinarizer()

# Transforming sex and embarked
X['sex'] = lb.fit_transform(X['sex'])
X['embarked'] = lb.fit_transform(X['embarked'])

X.head(5)
```

[50]:

	pclass	sex	age	embarked
1	1	0	38.0	1
3	1	0	35.0	0
6	1	1	54.0	0
10	3	0	4.0	0

```
11          1    0   58.0          0
```

Une autre méthode de transformation des features catégoriels, on peut convertir chaque catégorie d'un feature catégoriel en un entier. C'est "ordinal encoding".

[51]:
```python
# Ordinal encoding
X_ordinal = df[['pclass', 'sex', 'age', 'embarked']].copy()

enc = preprocessing.OrdinalEncoder()
X_ordinal[['sex', 'embarked']] = enc.fit_transform(X_ordinal[['sex',␣
↪'embarked']])

X_ordinal.head(5)
```

[51]:

	pclass	sex	age	embarked
1	1	0.0	38.0	0.0
3	1	0.0	35.0	2.0
6	1	1.0	54.0	2.0
10	3	0.0	4.0	2.0
11	1	0.0	58.0	2.0

Une autre méthode est "one-hot encoding". L'idée est de créer un nouveau feature pour chaque catégorie. La valeur du nouveau feature peut être 1 ou 0 qui représentent vrai et faux, ce qui indique si l'échantillon était dans cette catégorie dans le feature précédent.

[52]:
```python
# One-hot encoding
X_onehot = df[['pclass', 'sex', 'age', 'embarked']].copy()

enc = preprocessing.OneHotEncoder()
encoded_features = pd.DataFrame(
    enc.fit_transform(X_onehot[['sex', 'embarked']]).toarray(),
    columns = ['sex_'+s for s in list(enc.categories_[0])] +
              ['embarked_'+e for e in list(enc.categories_[1])],
    index = X_onehot.index
)

X_onehot = pd.concat([X_onehot, encoded_features], axis='columns')
X_onehot = X_onehot.drop(['sex', 'embarked'], axis='columns')

X_onehot.head(5)
```

[52]:

	pclass	age	sex_female	sex_male	embarked_C	embarked_Q	embarked_S
1	1	38.0	1.0	0.0	1.0	0.0	0.0
3	1	35.0	1.0	0.0	0.0	0.0	1.0
6	1	54.0	0.0	1.0	0.0	0.0	1.0
10	3	4.0	1.0	0.0	0.0	0.0	1.0
11	1	58.0	1.0	0.0	0.0	0.0	1.0

5. Importer des librairies Sklearn pour la modélisation, et faire le split train et test

La base de test est juste 10% des données, ce n'est pas suffisante. On a besoin que la base de test soit suffisamment grande pour représenter la base de données. Les tailles de base de test les plus courantes sont 20% et 25%. Ici, on prend 20% puisque la base de données est assez petite.

[53]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix

y = df['survived']

# Split train-test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

6. Créer une fonction pour imprimer les performances des modèles à construire

[54]:
```
def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    # Print the accuracy score, classification report and confusion matrix of␣
↪classifier
    if train:
        # Training performance
        print("Train result:\n")
        print("Accuracy score: {0:.4f}\n".format(accuracy_score(y_train, clf.
↪predict(X_train))))
        print("Classification report: \n {}\n".
↪format(classification_report(y_train, clf.predict(X_train))))
        print("Confustion matrix: \n {}\n".format(confusion_matrix(y_train, clf.
↪predict(X_train))))

        res = cross_val_score(clf, X_train, y_train, cv=10, scoring='accuracy')
        print("Average accuracy: \t {0:.4f}".format(np.mean(res)))
        print("Accuracy SD: \t\t {0:.4f}".format(np.std(res)))

    elif train == False:
        # Testing performance
        print("Test result:\n")
        print("Accuracy score: {0:.4f}\n".format(accuracy_score(y_test, clf.
↪predict(X_test))))
        print("Classification report: \n {}\n".
↪format(classification_report(y_test, clf.predict(X_test))))
        print("Confustion matrix: \n {}\n".format(confusion_matrix(y_test, clf.
↪predict(X_test))))
```

7. Définir et entrainer un Random Forest avec des paramètres par défaut.

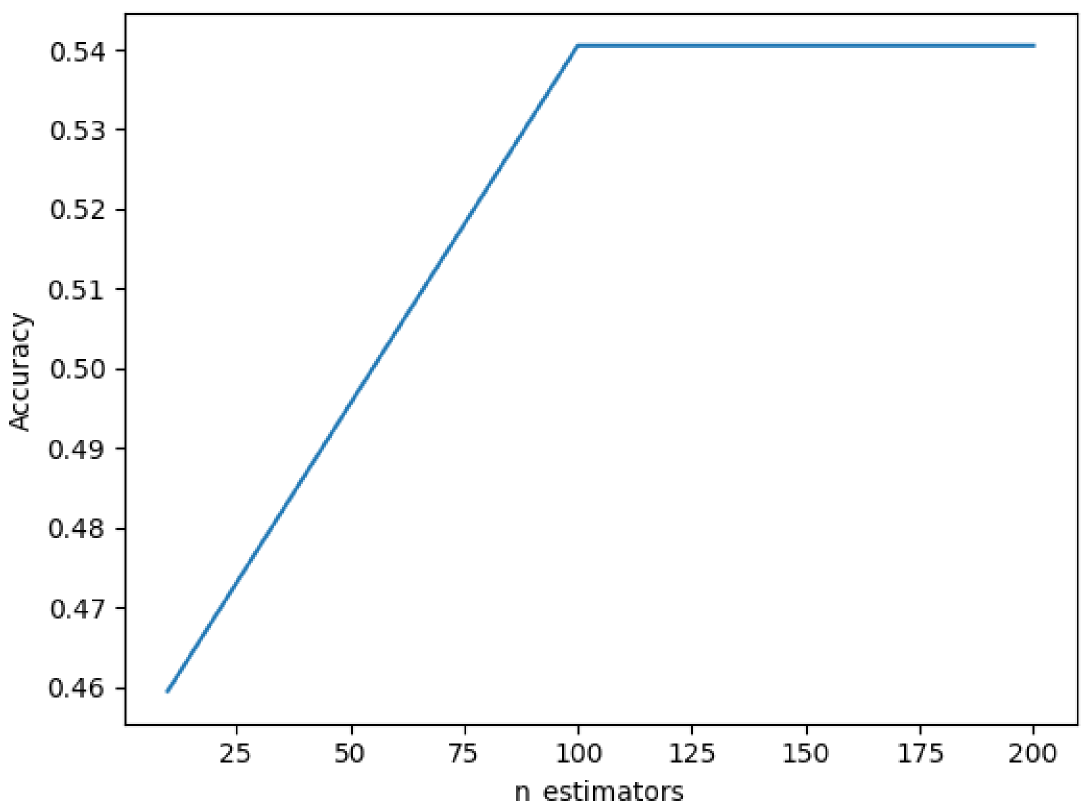
[55]:
```
# Defining the model
rf_clf = RandomForestClassifier(random_state=42, n_estimators=50)
```

```python
# Fitting
rf_clf.fit(X_train, y_train)
```

[55]: RandomForestClassifier(n_estimators=50, random_state=42)

Jouer avec l'hyperparamètre `n_estimators` avec des valeurs 10, 100 et 200 arbres. L'accuracy est le suivant:

[56]:
```python
accuracy_list = []
for n in [10, 100, 200]:
    rf_clf_ = RandomForestClassifier(random_state=42, n_estimators=n)
    rf_clf_.fit(X_train, y_train)
    accuracy_list.append(
        accuracy_score(y_test, rf_clf_.predict(X_test))
    )
plt.plot([10, 100, 200], accuracy_list)
plt.xlabel("n_estimators")
plt.ylabel("Accuracy")
plt.show()
```



8. Imprimer les performances en train et test

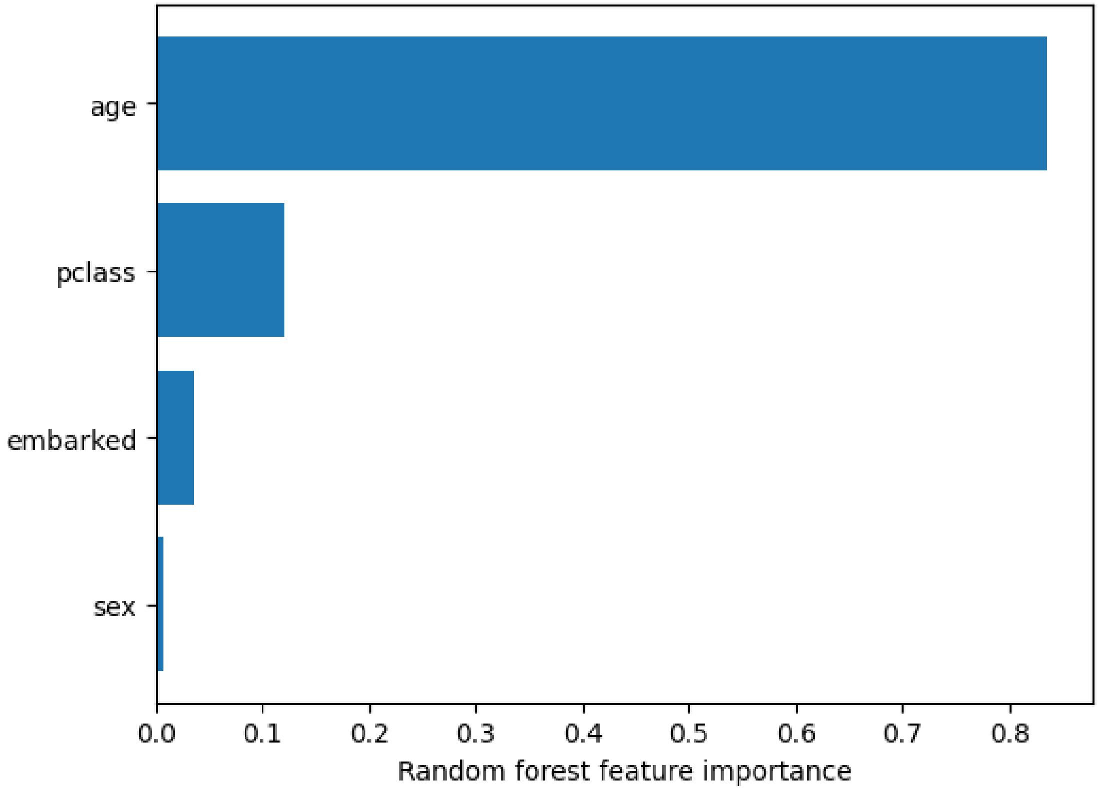
[57]:
```python
# Performance en train
print_score(rf_clf, X_train, y_train, X_test, y_test, train=True)

# Performance en test
print_score(rf_clf, X_train, y_train, X_test, y_test, train=False)
```

```
Train result:

Accuracy score: 0.8699

Classification report:
               precision    recall  f1-score   support

           0       0.86      0.74      0.80        50
           1       0.87      0.94      0.90        96

    accuracy                           0.87       146
   macro avg       0.87      0.84      0.85       146
weighted avg       0.87      0.87      0.87       146


Confustion matrix:
 [[37 13]
 [ 6 90]]

Average accuracy:        0.5971
Accuracy SD:             0.1083
Test result:

Accuracy score: 0.5676

Classification report:
               precision    recall  f1-score   support

           0       0.29      0.40      0.33        10
           1       0.74      0.63      0.68        27

    accuracy                           0.57        37
   macro avg       0.51      0.51      0.51        37
weighted avg       0.62      0.57      0.59        37


Confustion matrix:
 [[ 4  6]
 [10 17]]
```

9. Visualiser l'importance des features

[58]:
```
sorted_idx = rf_clf.feature_importances_.argsort()
plt.barh(X_train.columns[sorted_idx], rf_clf.feature_importances_[sorted_idx])
plt.xlabel("Random forest feature importance")
```

[58]: Text(0.5, 0, 'Random forest feature importance')



Exercice 2 : Grid search

1. Importer les librairies Sklearn

[59]:
```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
```

2. Définir un nouveau Random Forest

[60]:
```
# Defining new random forest model
rf_clf = RandomForestClassifier(random_state=42)
```

3. Déclarer les hyperparamètres à optimiser

[61]:
```
# Parameters to optimize
params_grid = {"max_depth": [1, 10],                # tree depth
```

```
               "min_samples_split": [2, 3, 10],      # the minimum number of␣
↪samples required to split an internal node
               "min_samples_leaf": [1, 3, 10],       # minimum number of samples␣
↪required to be at a leaf node
               "bootstrap": [True, False],           # bootstrap samples are used␣
↪when building trees
               "criterion": ['gini', 'entropy'],
               "n_estimators": [10, 20, 30]}         # splitting criteria
```

4. Définir le gridsearch et lancer l'entrainement

[62]:
```
# Defining grid search from sklearn
grid_search = GridSearchCV(rf_clf, params_grid,
                           n_jobs=-1, cv=5,
                           verbose=1, scoring='accuracy')
grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
```

[62]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=42), n_jobs=-1,
             param_grid={'bootstrap': [True, False],
                         'criterion': ['gini', 'entropy'], 'max_depth': [1, 10],
                         'min_samples_leaf': [1, 3, 10],
                         'min_samples_split': [2, 3, 10],
                         'n_estimators': [10, 20, 30]},
             scoring='accuracy', verbose=1)
```

Le paramètre `scoring` peut être `precision`, `recall`, etc.

5. Imprimer le meilleur score et le meilleur modèle

[63]:
```
grid_search.best_score_

grid_search.best_estimator_.get_params()
```

[63]:
```
{'bootstrap': False,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'entropy',
 'max_depth': 10,
 'max_features': 'sqrt',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 3,
 'min_samples_split': 10,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 30,
 'n_jobs': None,
```

```
'oob_score': False,
'random_state': 42,
'verbose': 0,
'warm_start': False}
```