



Programming assignment 2: Random forests, DAT340/DIT867, Applied ML

- Alfredo Serafini
- Erling Hjermland
 - Group PA2-10

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction import DictVectorizer
```

```
train = pd.read_csv("adult_train.csv")
test = pd.read_csv("adult_test.csv")
```

test

	<div>age int64</div> <div>17 - 90</div> <div></div>	<div>workclass object</div> <div>Private 68.9% Self-emp-n... 8.1% 7 others 23%</div>	<div>education object</div> <div>HS-grad 32.4% Some-colle... .. 22% 14 others 45.5%</div>	<div>education-num ...</div> <div>1 - 16</div> <div></div>	<div>marital-status o</div> <div>Married-c... 45.5% Never-mar... 33.4% 5 others 21.2%</div>	<div>occupation obje...</div> <div>Prof-spec... 12.5% Exec-mana... 12.4% 13 others 75.1%</div>	<div>relationship o...</div>	<div>race object</div>
0	25	Private	11th	7	Never-married	Machine-op-inspct	Own-child	Black
1	38	Private	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White
2	28	Local-gov	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White
3	44	Private	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black
4	18	?	Some-college	10	Never-married	?	Own-child	White
5	34	Private	10th	6	Never-married	Other-service	Not-in-family	White
6	29	?	HS-grad	9	Never-married	?	Unmarried	Black
7	63	Self-emp-not-inc	Prof-school	15	Married-civ-spouse	Prof-specialty	Husband	White
8	24	Private	Some-college	10	Never-married	Other-service	Unmarried	White
9	55	Private	7th-8th	4	Married-civ-spouse	Craft-repair	Husband	White

```
# basic preprocessing stuff
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction import DictVectorizer

x_train = train.drop("target", axis=1)
y_train = train["target"]

x_test = test.drop("target", axis=1)
y_test = test["target"]

x_train_dict = x_train.to_dict('records')
x_test_dict = x_test.to_dict('records')

preprocessing_pipeline = make_pipeline(DictVectorizer(), StandardScaler(with_mean=False)) #disabling standardscaler makes the output more understandable

x_train = preprocessing_pipeline.fit_transform(x_train_dict)
x_test = preprocessing_pipeline.transform(x_test_dict)
```

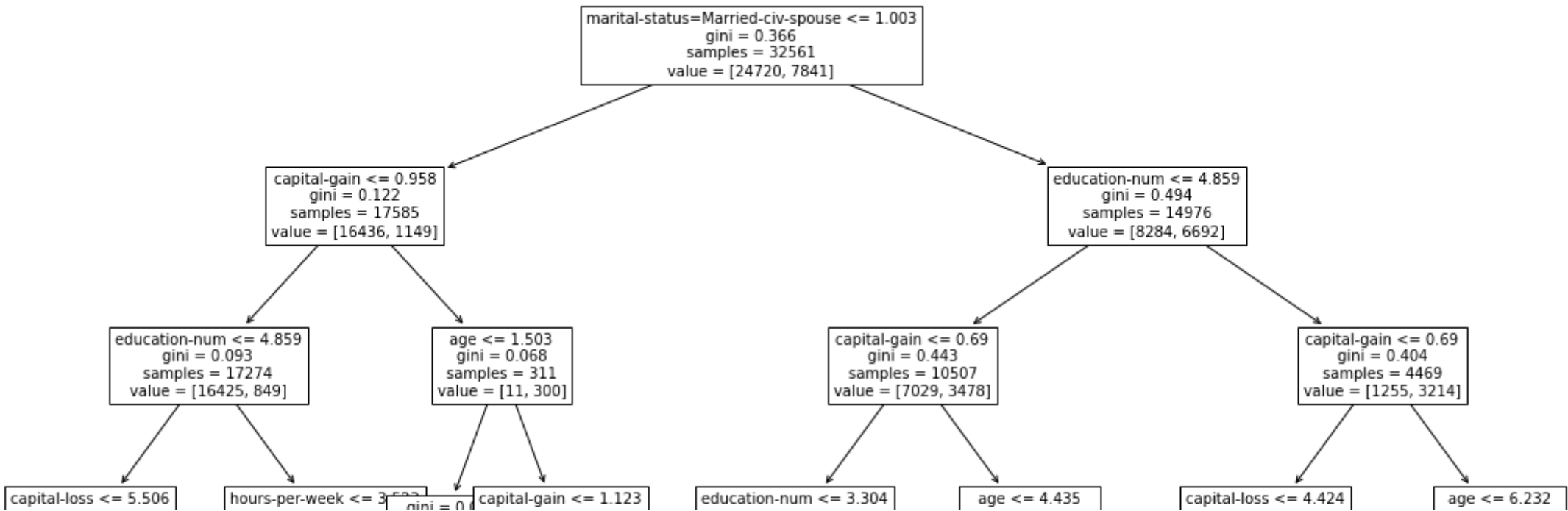
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt

clf = DecisionTreeClassifier(max_depth=4)
print("Mean cross-val accuracy:", np.mean(cross_val_score(clf, x_train, y_train))*100, "%")

clf.fit(x_train, y_train)
plt.figure(figsize=(20,10))
plot_tree(clf, feature_names=preprocessing_pipeline.get_feature_names_out(), fontsize=10);

#could also include the Decision tree in the pipeline, however we found easier to handle with the classifier outside.
#Had there been a longer pipeline it would have been another deal
```

Mean cross-val accuracy: 84.43845110761279 %



```
# for evaluation
from sklearn.metrics import accuracy_score

# a few different types of classifiers
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
# turn off annoying warnings
import warnings; warnings.simplefilter('ignore')

# and the VotingClassifier
from sklearn.ensemble import VotingClassifier
```

```
ensemble = [
    ('lr', LogisticRegression()),
    ('dt', DecisionTreeClassifier(max_depth=3)),
    ('lr1', LogisticRegression(penalty='l1', solver='liblinear')),
    ('mlp', MLPClassifier(hidden_layer_sizes=(8), max_iter=100))
]
```

```
voting = VotingClassifier(ensemble)
#voting = VotingClassifier(ensemble, voting='soft', n_jobs=-1, verbose=True) # hard is default
#VotingClassifier(estimators, *, voting='hard', weights=None, n_jobs=-1, flatten_transform=True, verbose=False)
```

```
voting.fit(x_train, y_train)
```

```
accuracy_score(y_test, voting.predict(x_test))
```

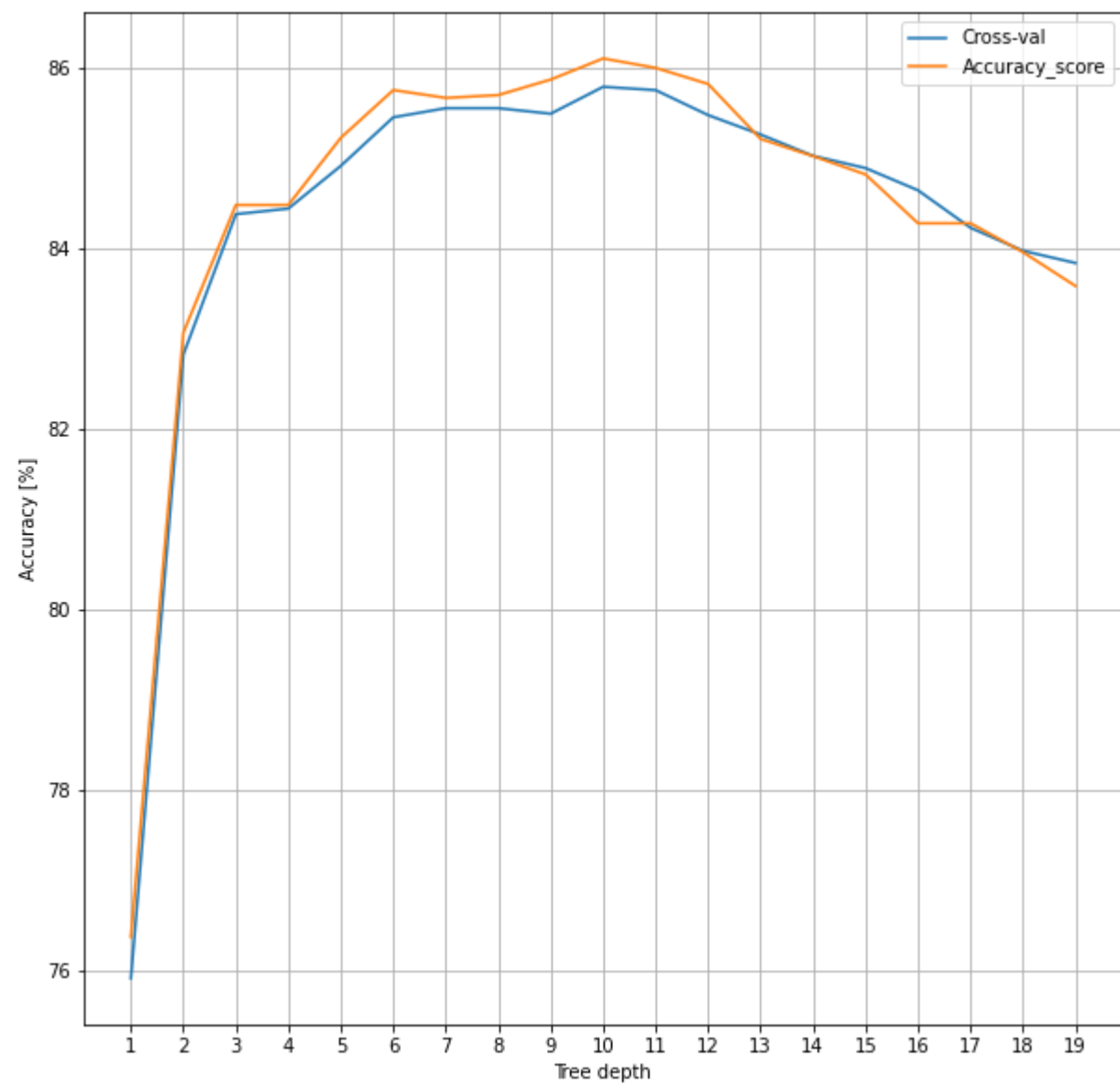
0.8530188563356059

Task 2 Decision trees and random forest

```
score = []
score2 = []
depth = list(range(1, 20))

for i in depth:
    clf = DecisionTreeClassifier(max_depth=i)
    score.append(100*np.mean(cross_val_score(clf, x_train, y_train, scoring="accuracy")))
    clf.fit(x_train, y_train)
    score2.append(100*accuracy_score(y_test, clf.predict(x_test)))
```

```
plt.figure(figsize=(10,10))
plt.xlabel("Tree depth")
plt.ylabel("Accuracy [%]")
plt.xticks(depth)
plt.grid()
plt.plot(depth, score)
plt.plot(depth, score2)
plt.legend(['Cross-val', 'Accuracy_score'])
plt.show()
```



The trees are clearly overfitting for depths above 10, and probably a bit before. $2^{10}=1024$ leaf nodes are many nodes, so perhaps a depth of 6 is optimal.

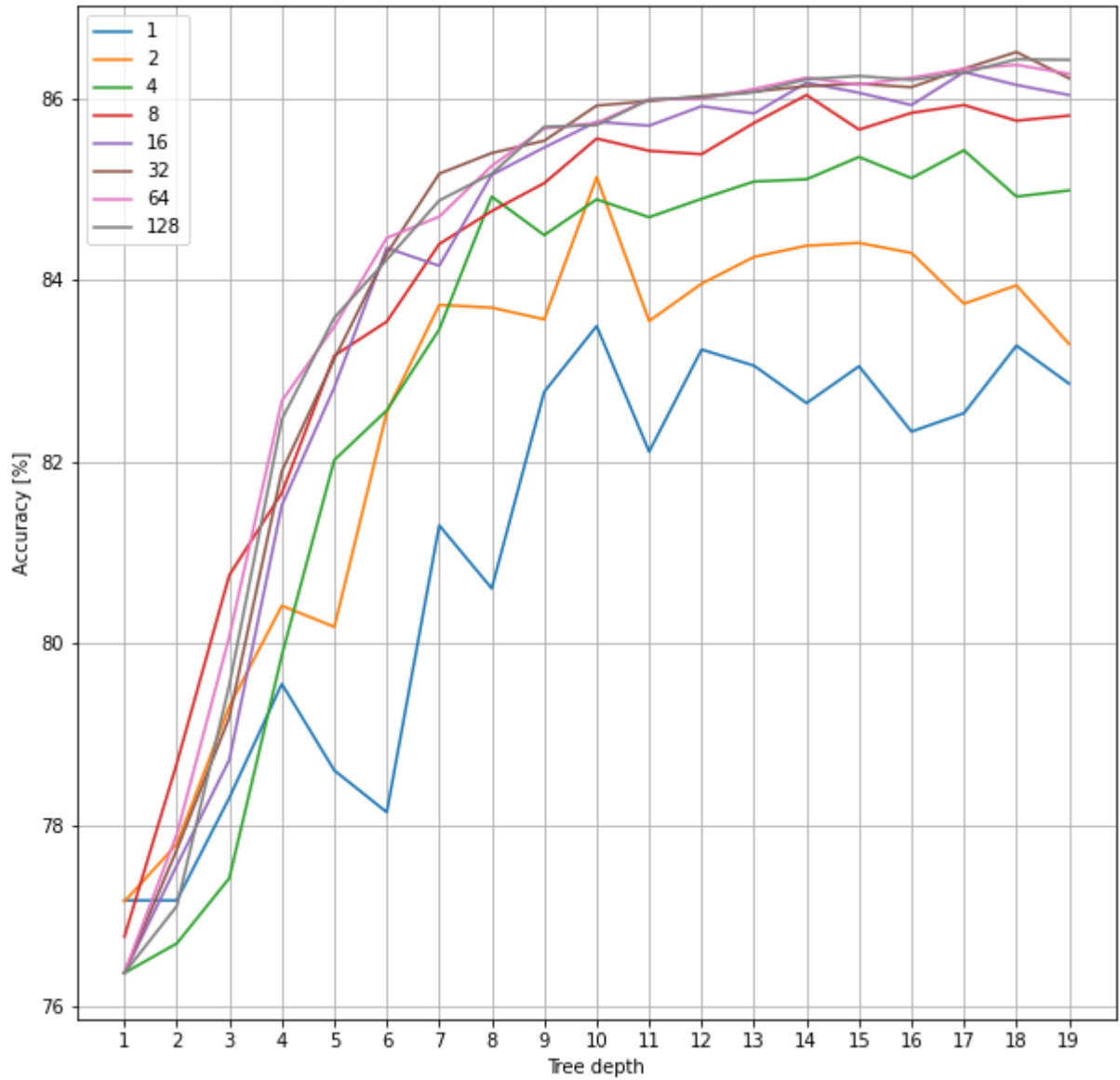
```
from sklearn.ensemble import RandomForestClassifier

depth = list(range(1, 20))
forest_size = [2**i for i in range(0, 8)]

tot_scores = []

for i in forest_size:
    score = []
    for d in depth:
        rf = RandomForestClassifier(n_estimators=i, max_depth=d, random_state=0, n_jobs=-1)
        rf.fit(x_train, y_train)
        score.append(100*accuracy_score(y_test, rf.predict(x_test)))
    tot_scores.append(score)

plt.figure(figsize=(10,10))
plt.xlabel("Tree depth")
plt.ylabel("Accuracy [%]")
plt.xticks(depth)
plt.grid()
for ts in tot_scores:
    plt.plot(depth, ts)
plt.legend(forest_size)
plt.show()
```



We can see that a huge number of trees in the forest does not improve the accuracy by much compared to around 16. 16 trees does almost as good of a job as 128 trees. With an ensable size of 1 the random disturbances in the training-data are not nulled out by the random disturbances in the other trees, as there is only one tree. Therefore, there are more random fluctuations in the accuracy-curve. The smaller ensambles show signs of overfitting, whilst the bigger ensambles are more resistant to this phenomenon. This is because of the small randomizing of data each tree in the ensambles are trained on making the total ensemble more robust against the variations of real-world data. For increasing depth the bigger ensambles converge towards an accuracy of around 86,5 %, but reach 86% at a depth of 10 making increasing depth unnecessary. Last but not least, with increasing ensemble size, one observe a better accuracy score at the expense of the computation time. In conclusion, we found that 16 trees are the right balance between efficiency and final accuracy value.

Task 3

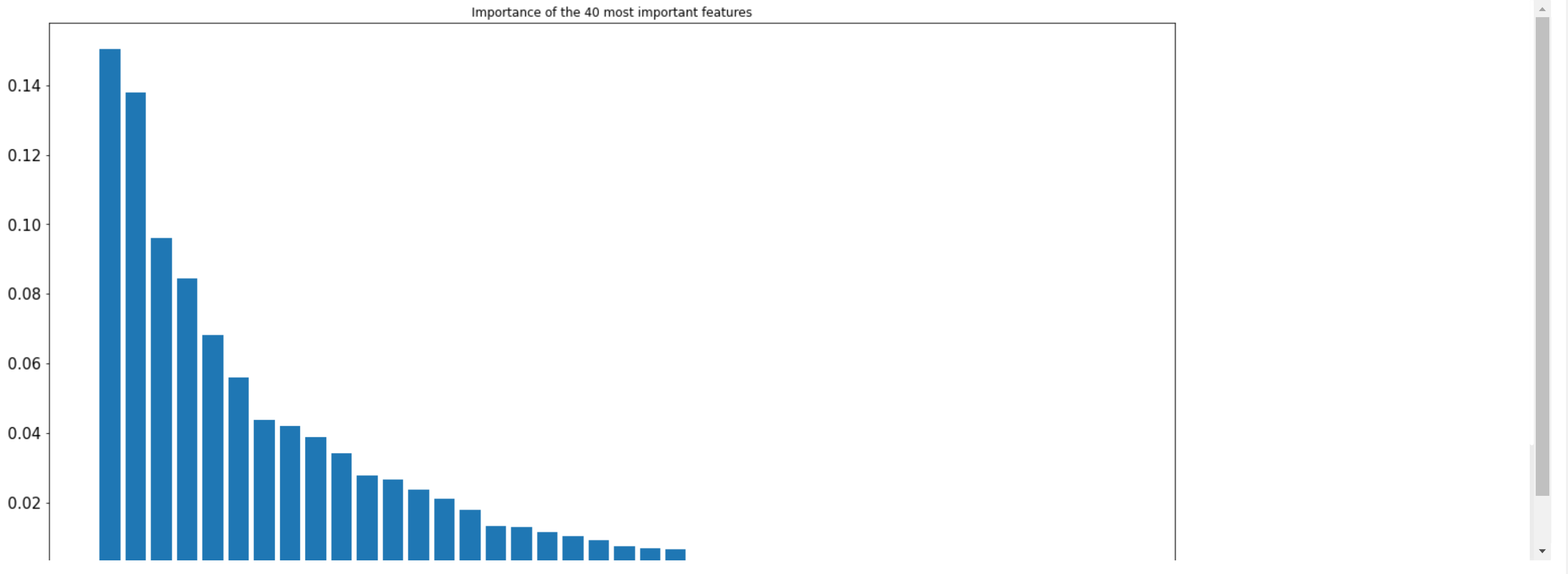
```
# rf.feature_importances_  
from sklearn.ensemble import RandomForestClassifier
```

```
rf = RandomForestClassifier(n_estimators=16, max_depth=10, n_jobs=-1)  
rf.fit(x_train, y_train)
```

```
feature_importance = rf.feature_importances_  
features = preprocessing_pipeline.get_feature_names_out()
```

```
sorting = feature_importance.argsort()[::-1]  
feature_importance = feature_importance[sorting]  
features = features[sorting]  
plt.figure(figsize=(20, 10))  
plt.yticks(fontsize=15)  
plt.xticks(rotation = 90, fontsize=15)  
plt.bar(features[:40], feature_importance[:40])  
plt.title("Importance of the 40 most important features")  
plt.show()
```





We noticed that the feature of being married is rather suspicious to be a better feature than capital gain to categorize the income of a subject. Marriage is expected to be common in both categories. More valuable feature should be the capital gain, real-estate or stock earnings, and education level. These properties are expected to discriminate between high- and low-income people.

As exemplified in [Beware Default Random Forest Importances - explained.ai](#), one possible remedy or alternative to obtain better feature selections is to employ what is called, *permutation importance*. *Permutation importance* is highly applicable to any model and is an efficient and reliable technique. What *permutation importance* technique does is shuffling stochastically each predictor variable and observe the effect on the model accuracy to estimate the variable importance. This technique is widely used due to its lack of internal model parameter dependencies, e.g. linear regression coefficients.

Below are 2 implementations of permutation-importances.

One can see that marital status is no longer the most important feature. First our implementation:

```
def permutation_importances(rf, X_train, y_train, metric):  
    baseline = metric(rf.predict(X_train), y_train)  
    imp = []  
    for col in range(X_train.shape[1]):  
        save = X_train[:, col].copy()  
        X_train[:, col] = X_train[:,col][np.random.permutation(X_train.shape[0])]   
        m = metric(rf.predict(X_train), y_train)  
        X_train[:, col] = save  
        imp.append(baseline - m)  
    return np.array(imp)
```

```
feature_importance = permutation_importances(rf, x_train, y_train, accuracy_score)
features = preprocessing_pipeline.get_feature_names_out()

sorting = feature_importance.argsort()[::-1]
feature_importance = feature_importance[sorting]
features = features[sorting]
plt.figure(figsize=(20, 10))
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.xticks(rotation = 90, fontsize=15)
plt.bar(features[:40], feature_importance[:40])
plt.title("Importance of the 40 most important features")
plt.show()
```



Implemented with Sklearn's function: (slow)

```
from sklearn.inspection import permutation_importance

r = permutation_importance(rf, x_train.toarray(), y_train,
                           n_repeats=15,
                           random_state=0,
                           n_jobs=-1)

sorting = r.importances_mean.argsort()[::-1]
r = r.importances_mean[sorting]
features = preprocessing_pipeline.get_feature_names_out()
features = features[sorting]
```



```
plt.figure(figsize=(20, 10))
plt.yticks(fontsize=15)
plt.xticks(rotation = 90, fontsize=15)
plt.bar(features[:40], r[:40])
plt.title("Importance of the 40 most important features")
plt.show()
```



These two latter plots show that our results are rather consistent, though using two distinct approaches such as custom function `permutation_importances` and `scikit-learn` library. In other words, we can see that for both scenarios we have *capital-gain* and *marital-status=Married-civ-spouse*, *education-num* and *capital loss*, the most compelling features of per-capita income.

Last, in accordance with [Beware Default Random Forest Importances - explained.ai](#), we found that some features such as the 5th and 6th one, respectively, are flipped between these two latter plots.