Machine Learning

Università Roma Tre
Dipartimento di Ingegneria

Anno Accademico 2021 - 2022

Esercitazione: Ensembles di Decision Trees (Ex 06)

Sommario

- Ensembles
- Random Forests
- Gradient boosted regression trees

Ensembles

- In ML, l'**ensembles** un approccio che combina più modelli di ML per creare un nuovo modello più sofisticato, che potenzialmente aggrega i benefici dei singoli modelli.
- Esistono vari approcci di ensembles. Due approcci basati sui decision trees (**DT**) si sono dimostrati molto adatti in vari domini:
  - Random forests
  - Gradient boosted decision trees.

Ensembles: Random forests

- I **random forests** (**RF**) sono una collezione di DTs, ognuno costruito in modo leggermo diverso dall'altro durante il training.
- I DTs tendono a mostrare overfitting. I RF tendono ad affrontare questa problematica: ogni albero può mostrare overfitting su certi dati, ma se ne costruiamo diversi in modo indipendente e mediamo i risultati complessivi, l'effetto dell'overfitting si riduce.
- Per creare diversi DTs, introduciamo un elemento casuale durante il processo di training, ad esempio selezionando:
  - diversi set di training
  - diverse features in ogni split test

Scikit-learn: Random forests

- In scikit-learn esiste una implementazione dei RF per la classificazione e per la regressione: *RandomForestClassifier* e *RandomForestRegressor*.
- Il numero di DTs è un iperparametro del modello RF e si imposta col parametro *n_estimators* del costruttore (es. 10).
- Inizialmente si costruisce un *bootstrap sample* dei dati.
  - Dal training set estraiamo *n_samples* istanze in modo casuale, con ripetizione, e ripetiamo n_samples volte.
  - Il dataset che si ottiene è grande come quello originale, ma alcune istanze si possono ripetere, altre sono mancanti (approssimativamente 1/3)
- Es.: se il dataset = ['a', 'b', 'c', 'd'], un possibile bootstrap è ['b', 'd', 'd', 'c'], un altro ['d', 'a', 'd', 'a'].
- Dopodiché si addestra un DT per ogni boostrap sample.

Scikit-learn: Random forests

- Un ulteriore elemento casuale è introdotto in ogni nodo dell'albero.
- Durante la costruzione, invece di scegliere il test migliore, si selezionando un modo casuale un sottoinsieme di features e si seleziona la migliore considerando tale sottoinsieme.
- Il numero di features è impostato col parametro del costruttore *max_features* (ulteriore iperparametro del modello).
  - Un valore alto di *max_features* riduce la casualità nel modello RF, ma migliora il fit sui dati. Un valore basso produce degli alberi molto complessi per raggiungere lo stesso livello di fit.
- Per generare l'output, ogni DT è valutato sull'istanza in input e i risultati sono sottoposti a *soft voting,* cioè le probabilità per ogni *label* ottenute dai singoli DT sono mediate e la classe con probabilità più alta è l'output del RF.

Scikit-learn: Random forests e two_moons

- **Esercizio**: col dataset *two_moons* crea un modello RF con 5 alberi.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=100, noise=0.25, random_state=3)
...
```

Scikit-learn: Random forests e two_moons

- Col dataset two_moons creiamo un modello RF con 5 alberi:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=100, noise=0.25, random_state=3)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
      random_state=42)

forest = RandomForestClassifier(n_estimators=5, random_state=2)
forest.fit(X_train, y_train)
```

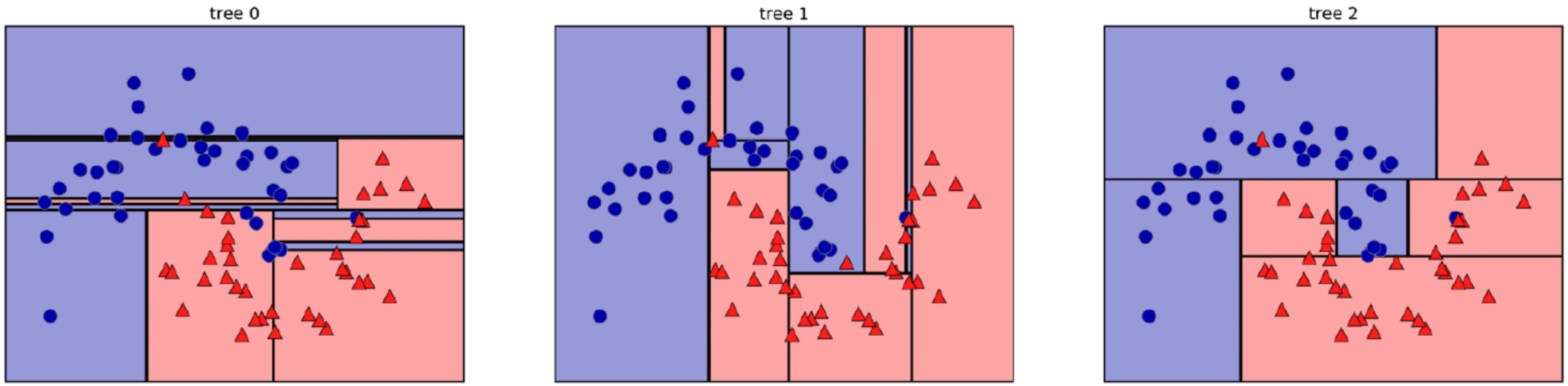
- I parametri sono salvati nella variabile *estimator_* del modello.

- Possiamo rappresentare i decision boundary per ogni modello:

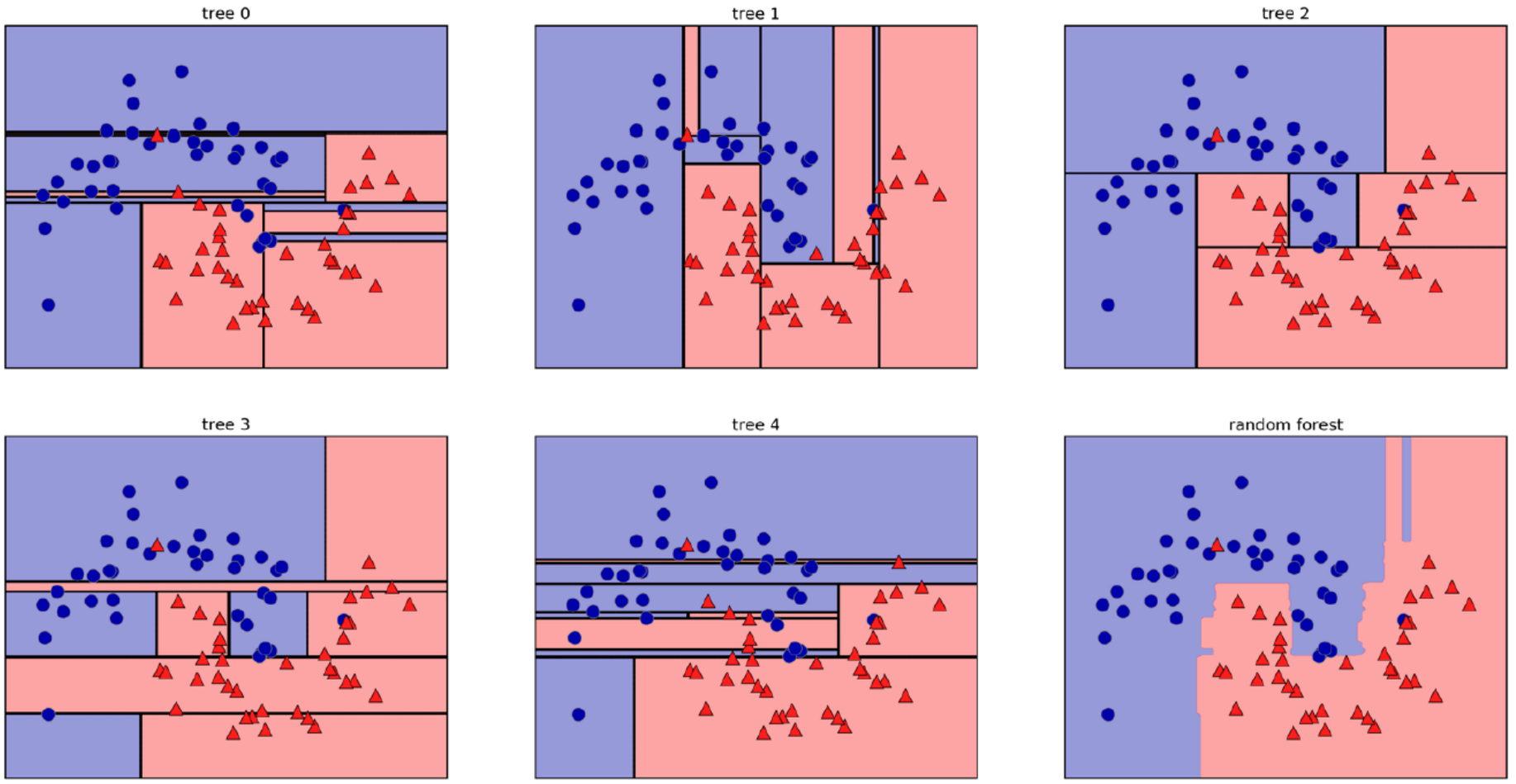
```
fig, axes = plt.subplots(2, 3, figsize=(20, 10))
for i, (ax, tree) in enumerate(zip(axes.ravel(), forest.estimators_)):
   ax.set_title("Tree {}".format(i))
   mglearn.plots.plot_tree_partition(X_train, y_train, tree, ax=ax)
mglearn.plots.plot_2d_separator(forest, X_train, fill=True, ax=axes[-1, -1],
      alpha=.4)
axes[-1, -1].set_title("Random Forest")
mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
```

Scikit-learn: Random forests e two_moons

- Cosa puoi notare riguardo i modelli e i training set?



Scikit-learn: Random forests e two_moons



- Ogni modello ha decision boundaries distinti, dove alcune istanze non sono correttamente classificati.
- Ogni modello ha un training set leggermente distinto: alcune istanze del training set complessivo non sono presenti.
- Le boundaries del modello finale sono più "smooth".

Scikit-learn: Random forests e breast cancer dataset

- **Esercizio**: crea un RF per il dataset breast cancer con 100 alberi, e valuta l'accuracy nel training e test set, confrontandola con quella ottenuta con un singolo DT.

Scikit-learn: Random forests e breast cancer dataset

- **Esercizio**: crea un RF per il dataset breast cancer con 100 alberi, e valuta l'accuracy nel training e test set, confrontandola con quella ottenuta con un singolo DT.

```
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)
forest = RandomForestClassifier(n_estimators=100, random_state=0)
forest.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(forest.score(X_train,
y_train)))
print("Accuracy on test set: {:.3f}".format(forest.score(X_test, y_test)))
```

```
Accuracy on training set: 1.000
Accuracy on test set: 0.972
```

- L'accuracy è più alta rispetto al modello lineare e al DT.
- È possibile fare un tuning con i parametri max_features e l'approccio pruning, ma su alcuni dataset i valori di default possono essere già sufficienti.

Scikit-learn: Random forests e breast cancer dataset

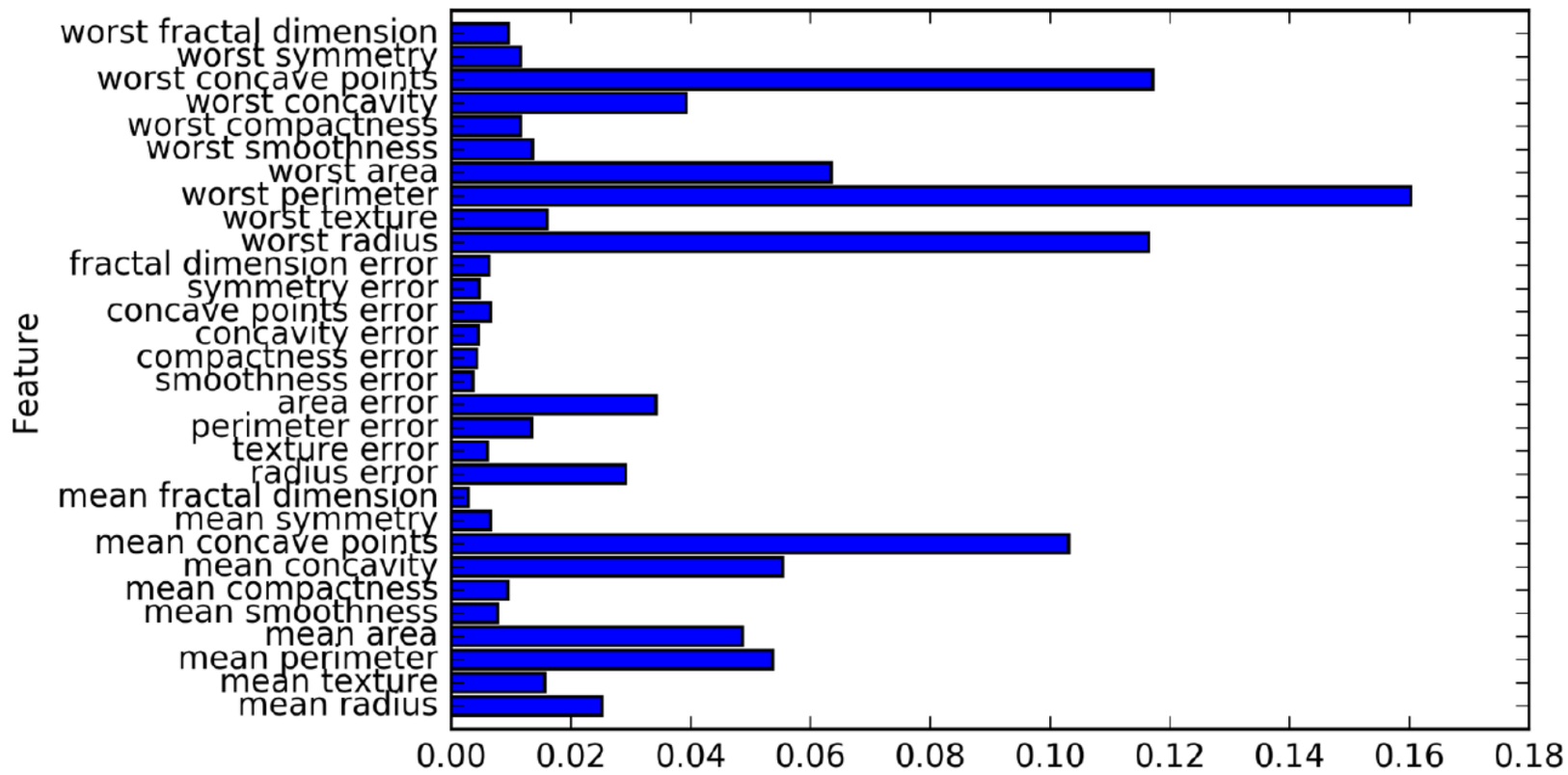
- Cosa ti aspetti dalla feature importance ottenuta mediando i valori dei singoli trees?

```
plot_feature_importances_cancer(forest)
```

Scikit-learn: Random forests e breast cancer dataset

- Cosa ti aspetti dalla feature importance ottenuta mediando i valori dei singoli trees?

```
plot_feature_importances_cancer(forest)
```



- Il valore aggregato ha più variabilità e tendenzialmente è più accurato. Il modello considera più features dando meno importanza alle singole (es. *worst radius*)

Considerazioni sui Random forests (1)

- I RF sono modelli di ML molto utilizzati essendo versatili, non richiedono lunghe fasi di tuning degli iperparametri e il rescaling dei dati.
- D'altro canto se hai bisogno di una rappresentazione compatta, il singolo DT è la soluzione migliore. È impossibile interpretare il valore di centinaia o più DT, soprattutto se hanno profondità elevate.
- Le implementazione dei RF possono essere facilmente parallelizzate su più CPU. Il parametro *n_jobs* imposta il numero di core da impiegare (un valore pari a -1 indica l'uso di tutti i core).
- L'approccio random nei RF rende i modelli generati sugli stessi dati anche molto diversi tra loro. Se vuoi ottenere risultati riproducibili, imposta il parametro *random_state*.
- I RF non mostrano buone prestazioni su dati sparsi e/o con molte features, es. dati testuali. I modelli lineare sono da preferire.

Considerazioni sui Random forests (2)

- Un parametro elevato di n_estimators solitamente migliora le performance, ma richiede più tempo e memoria per il training.
- Una indicazione per il parametro *max_features* è impostarlo pari a *sqrt(n_features)* per la classificazione, e *log2(n_features)* per la regressione.

Ensembles: Gradient boosted regression trees

- I *Gradient boosted regression trees* (*gradient boosting machines*) **GBRT** sono un approccio di *ensembles*, e possono essere impiegate sia per la classificazione sia per la regressione.
- A differenza dei RF, gli alberi sono costruiti in modo sequenziale, dove ogni albero tenta di risolvere i problemi mostrati in precedenza. L'algoritmo è basato sull'approccio *boosting* visto in precedenza.
- Al posto dell'elemento casuale, è impiegato l'approccio pre-pruning. Gli alberi prodotti non sono profondi (tipicamenti depth da 1 a 5), e questo rende il modello più compatto e veloce nelle predizioni.
- I singoli alberi sono modelli *semplici* (in ML sono spesso chiamati *weak learners*) che producono buone performance su alcune istanze dei dati.
- Rispetto ai RF sono più sensibili alla scelta degli iperparametri, ma possono produrre risultati migliori, per questo sono spesso impiegati in scenari reali.

Ensembles: Gradient boosted regression trees

- Oltre al pre-pruning e al numero di alberi (*n_estimators*), un altro iperparametro fondamentale è il *learning_rate,* che controlla quanto un albero deve correggere gli errori prodotti dal precedente. Un valore elevato genera modelli più complessi. Allo stesso modo, un valore elevato di *n_estimators* incrementa la complessità e può ridurre gli errori commessi.

- In scikit-learn, la classe *GradientBoostingClassifier* implementa gli GBRT.

- Nel caso del Breast cancer dataset, con 100 alberi, con profondità max pari a 3 e un learning rate pari a 0.1:

```
from sklearn.ensemble import GradientBoostingClassifier
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)
gbrt = GradientBoostingClassifier(random_state=0)
gbrt.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(gbrt.score(X_train,
y_train)))
print("Accuracy on test set: {:.3f}".format(gbrt.score(X_test, y_test)))

Accuracy on training set: 1.000
Accuracy on test set: 0.958
```

Gradient boosted regression trees

- Otteniamo una accuracy pari al 100%, potrebbe indicare un possibile overfitting.
- **Esercizio**: prova ad incrementare il pre-pruning o ridurre il learning rate.

Gradient boosted regression trees

- **Esercizio**: prova ad incrementare il pre-pruning o ridurre il learning rate.

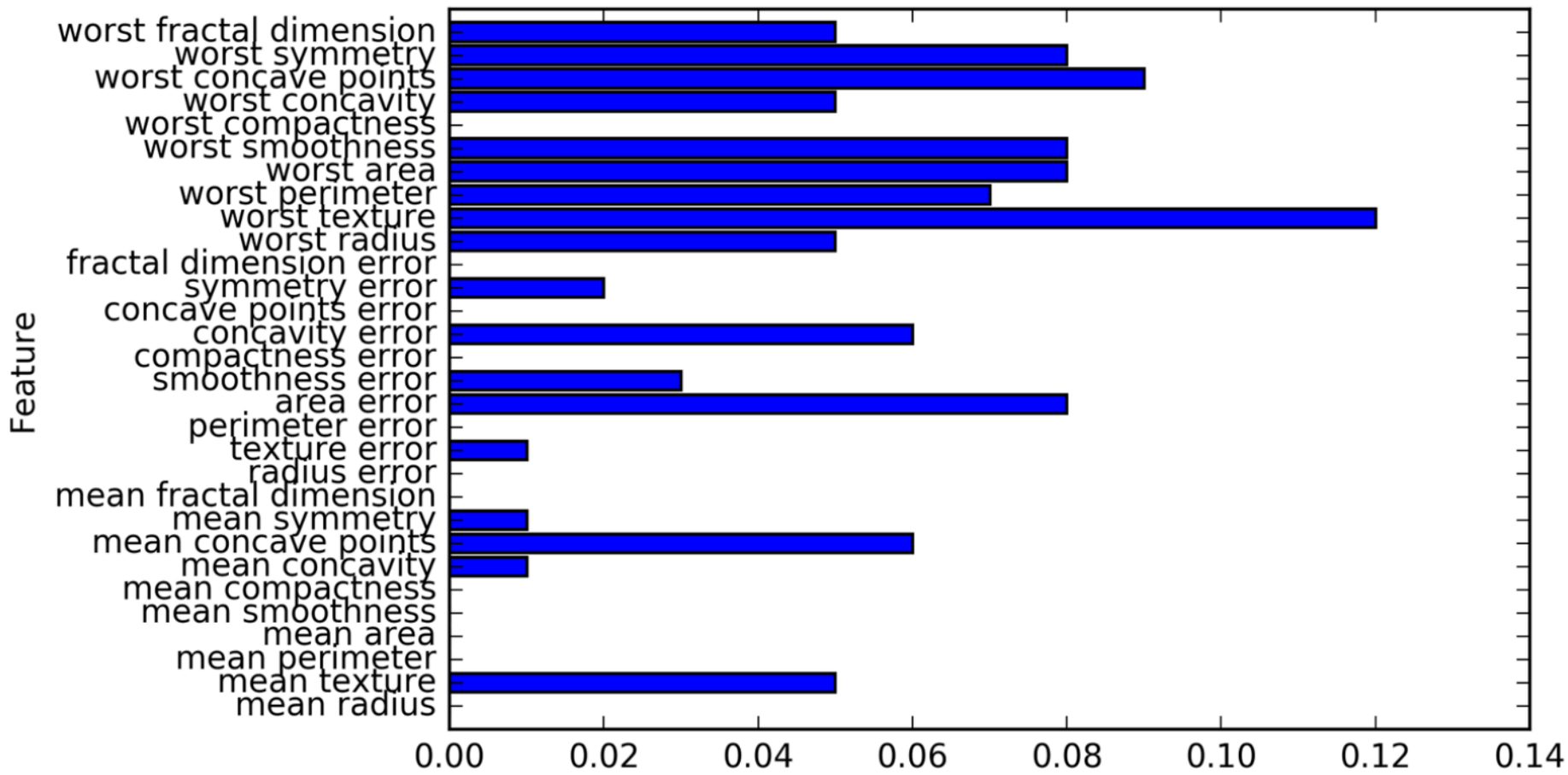
```
gbrt = GradientBoostingClassifier(random_state=0, max_depth=1)
gbrt.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(gbrt.score(X_train,
   y_train)))
print("Accuracy on test set: {:.3f}".format(gbrt.score(X_test, y_test)))

Accuracy on training set: 0.991
Accuracy on test set: 0.972

gbrt = GradientBoostingClassifier(random_state=0, learning_rate=0.01)
gbrt.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(gbrt.score(X_train,
     y_train)))
print("Accuracy on test set: {:.3f}".format(gbrt.score(X_test, y_test)))

Accuracy on training set: 0.988
Accuracy on test set: 0.965
```

- Entrambi gli approcci riducono la complessità e l'accuracy sul training set.
- In questo scenario, ridurre la profondità migliora maggiormente le performance.

Gradient boosted regression trees

- Avendo impiegato 100 alberi, è poco pratico visualizzare le decision boundaries di ognuno, ma possiamo analizzare le feature importance.

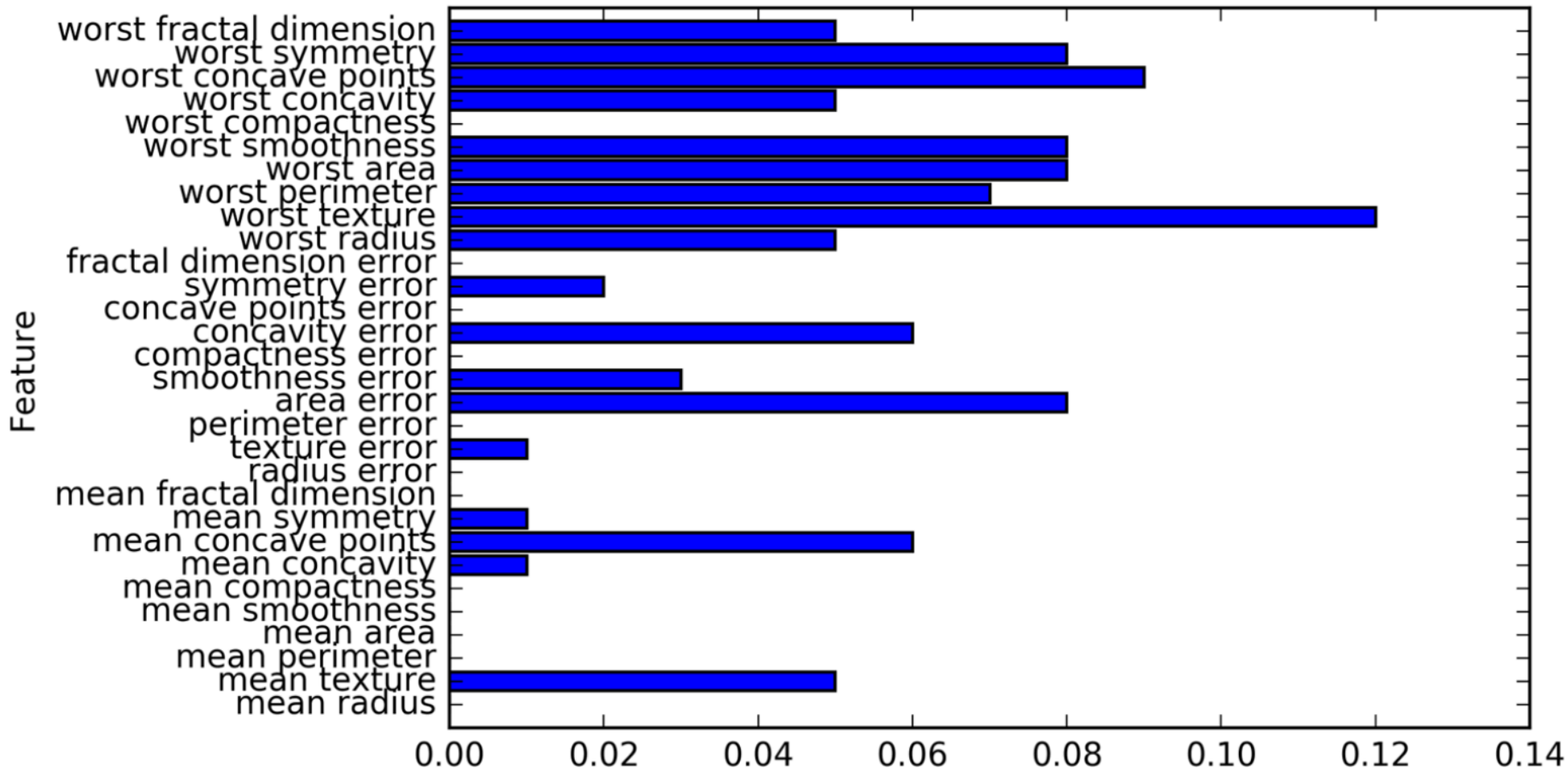
```
gbrt = GradientBoostingClassifier(random_state=0, max_depth=1)
gbrt.fit(X_train, y_train)
plot_feature_importances_cancer(gbrt)
```



- Noti differenze rispetto ai RF?

Gradient boosted regression trees

- I generale otteniamo *importance* simili, ma in questo caso alcune features hanno peso pari a 0, cioè sono completamente ignorate dal modello.



Gradient boosted regression trees: considerazioni (1)

- Entrambi gli approcci ensembles mostrano buoni risultati su dati simili. Si può applicare prima l'approccio RF, piuttosto robusto.
- I GBRT richiedono un tuning degli iperparametri più lungo rispetto ai RF.
- Se il tempo impiegato per la predizione non è soddisfacente, o è fondamentale raggiungere una accuracy massima, si può considerare il GBRT.
- Come per i RF, i GBRT funzionano bene senza rescaling, e su combinazioni di feature binary o continous. Ma non sono efficienti per dataset con molte features.
- I due iperparametri fondamentali sono *n_estimators* e *learning_rate*. Sono dipendenti l'uno dall'altro. Un basso learning rate richiede più alberi per raggiungere la stessa complessità. Un valore elevato di *n_estimators* migliora il modello, ma fa tendere il modello all'overfitting.
- Tipicamente si imposta *n_estimators* in base alle risorse a disposizione, dopodiché si ottimizza il valore *learning_rates*.

Gradient boosted regression trees: considerazioni (2)

- Altro iperparametro fondamentale è *max_depth* (o alternativamente *max_leaf_nodes*) per ridurre la complessità per ogni albero. Tipicamente si imposta a un valore molto basso, es. < 5.

- Con dataset di larghe dimensioni, si può considerare anche la libreria *xgboost*, che possiede una implementazione più ottimizzata.

Esercizio su ensembles

- **Esercizio**: impiegare i due approcci ensembles sui restanti dataset introdotti nelle precedenti esercitazioni:
  - *Forge dataset* (classificazione)
  - *wave dataset* (regressione)
  - *Boston housing dataset* (regressione)

- Valutare la accuracy rispetto all'approccio basato sulla regressione lineare e al singolo decision tree.
- Operare un tuning degli iperparametri per incrementare le performance.

Testi di Riferimento

- Andreas C. Müller, Sarah Guido. *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media 2016
- Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media 2017