Machine Learning

Università Roma Tre
Dipartimento di Ingegneria

Anno Accademico 2021 - 2022

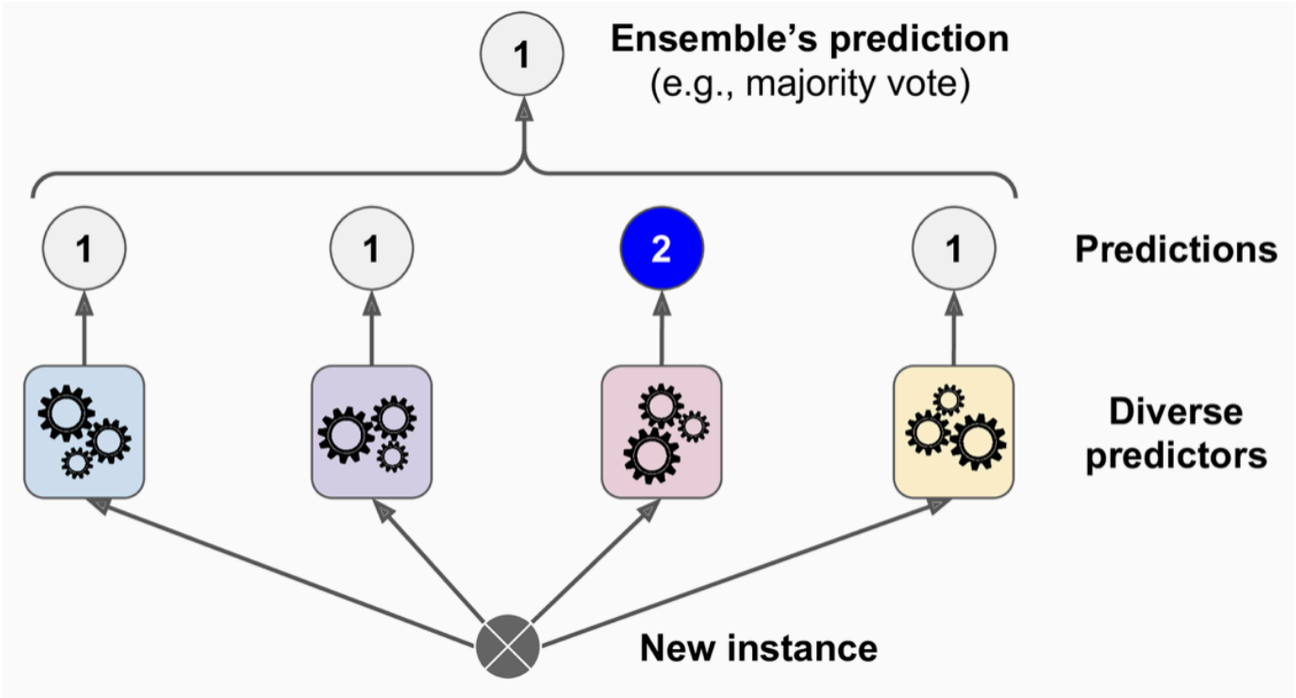
Esercitazione: Voting e Stacking ensembles (Ex 07)

Sommario

- Voting
- Stacking
- Mutilayer Stacking
- Datasets MNIST e notMNIST
- Altri dataset di immagini
- Esercitazioni

Ensembles: Voting

- L'approccio voting si ispira alla filosofia *wisdom of the crowd*. Supponiamo di avere più classificatori (es. Logistic regression, SVM, Random forest, k-NN). Prendiamo la predizione di ognuno e scegliamo quella che riceve "più voti". Questa forma di aggregazione prende il nome di *hard-voting*.



- Se partiamo da weak classifiers con accuracy non soddisfacente, il classificatore risultante può raggiungere accuracy elevate.

Scikit-learn: Voting

- La classe **VotingClassifier** di scikit-learn implementa l'approccio.
- **Esercizio**: completa il seguente frammento di codice basandoti sulla documentazione online di VotingClassifier.

```
from sklearn.ensemble import VotingClassifier

(... importa gli altri classificatori ...)

from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=500, noise=0.30, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

...

voting_clf = VotingClassifier(
    ...,
    voting='hard')

voting_clf.fit(X_train, y_train)
```

Scikit-learn: Voting

- Impieghiamo SVM, RandomForest e LogisticRegression:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=500, noise=0.30, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

```
log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC()

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

voting_clf.fit(X_train, y_train)
```

Scikit-learn: Voting

(segue)

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

```
LogisticRegression 0.864
RandomForestClassifier 0.896
SVC 0.888
VotingClassifier 0.904
```

Scikit-learn: Voting

- Se i classificatori impiegati sono in grado di stimare probabilità di appartenenza alle singole label, cioè implementano la funzione predict_proba(), il voting può valutare le medie delle probabilità prodotte da ogni classificatore.
- L'approccio si chiama *soft voting,* e si seleziona col parametro voting del costruttore:

```
voting='soft'
```

- **Esercizio**: controlla che i classificatori impiegati in precedenza implementino predict_proba() e, in caso affermativo, lancia nuovamente il codice precedente e valuta la differenza di performance.

Scikit-learn: Voting

```
og_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)
```

```
VotingClassifier(estimators=[('lr', LogisticRegression(random_state=42)),
                             ('rf', RandomForestClassifier(random_state=42)),
                             ('svc', SVC(probability=True, random_state=42))],
                 voting='soft')
```

```
from sklearn.metrics import accuracy_score

for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
```

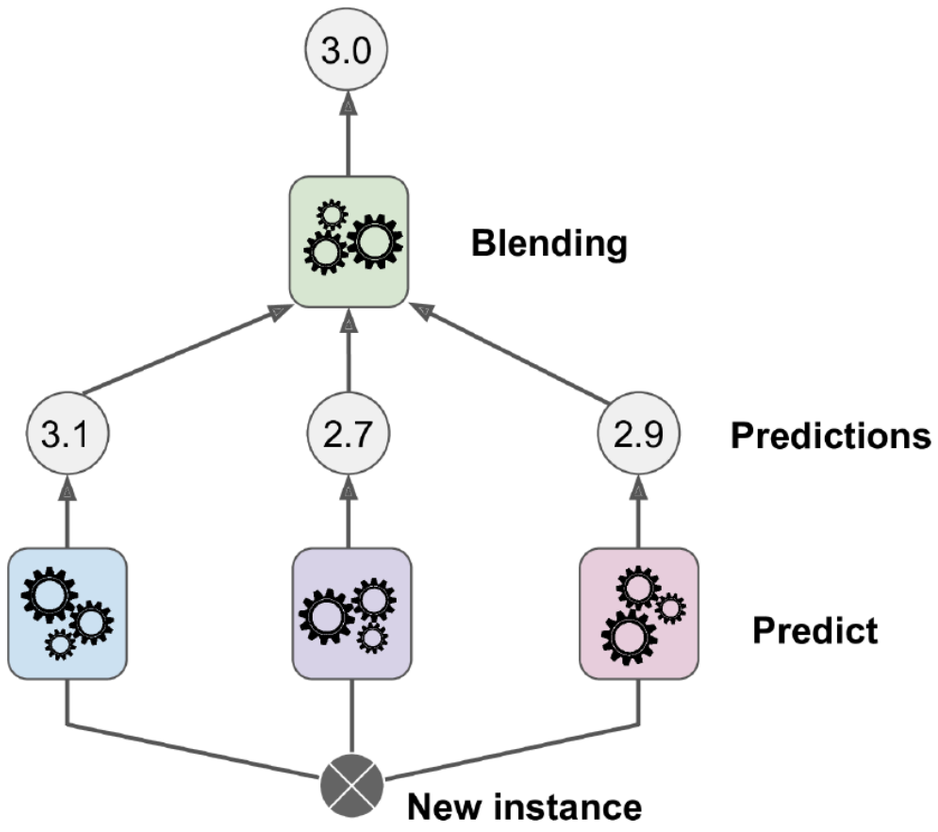
```
from sklearn.metrics import accuracy_score

for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
```

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

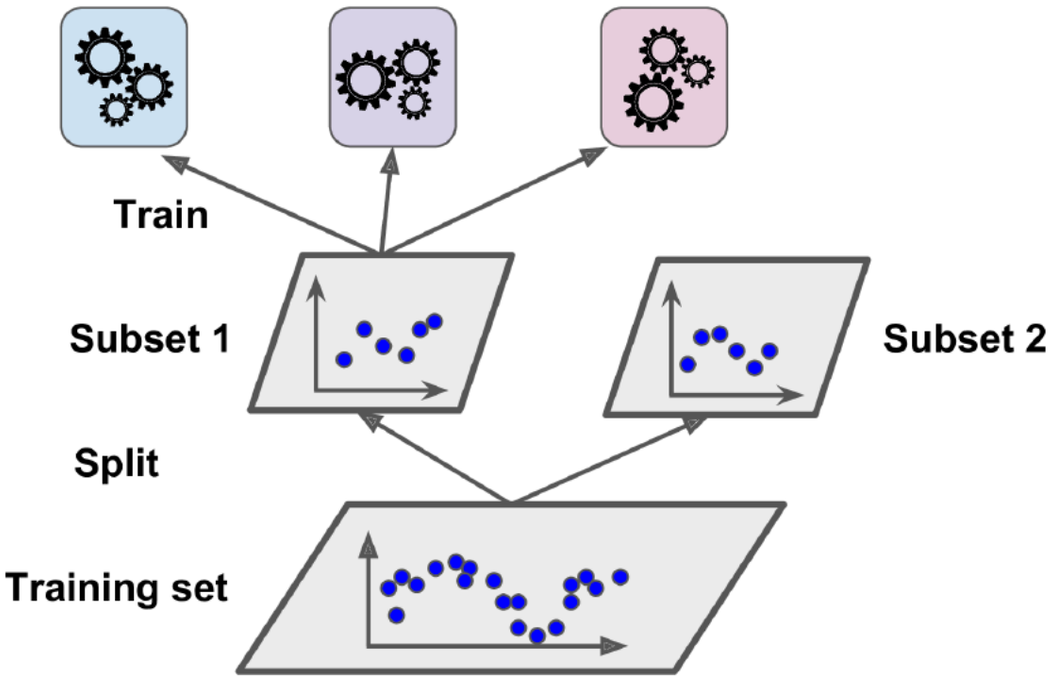
Ensembles: Stacking

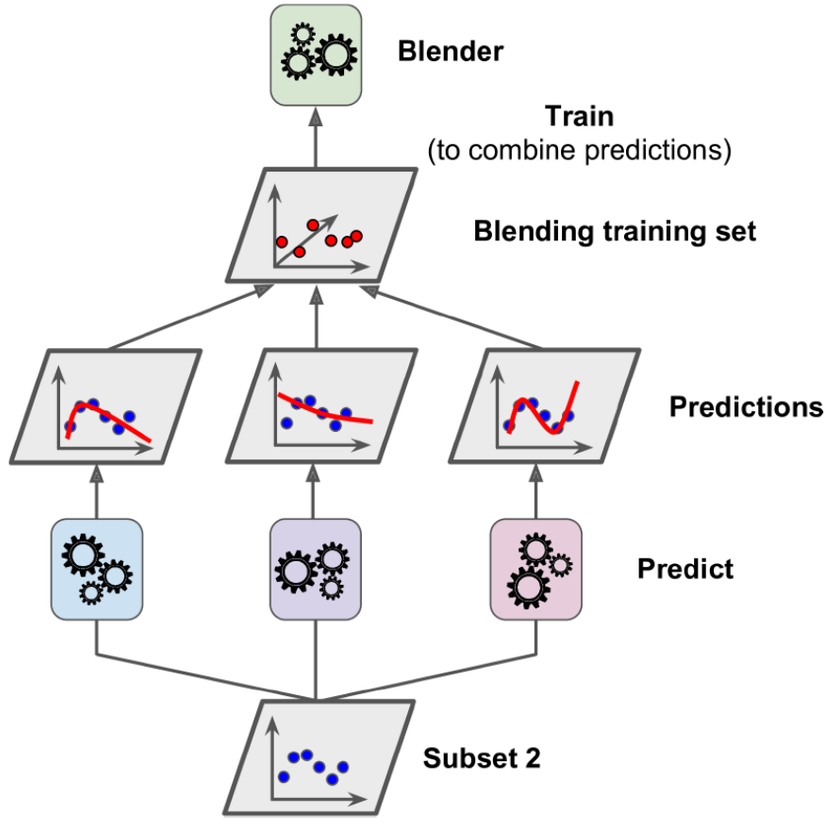
- Un ulteriore approcio ensembles è lo **stacking**, che sta per *stacked generalization*.
- Invece di aggregare il risultato con una tecnica di voting, addestriamo un ulteriore modello per questo scopo, chiamato *blender* o *meta learner*.



Stacking

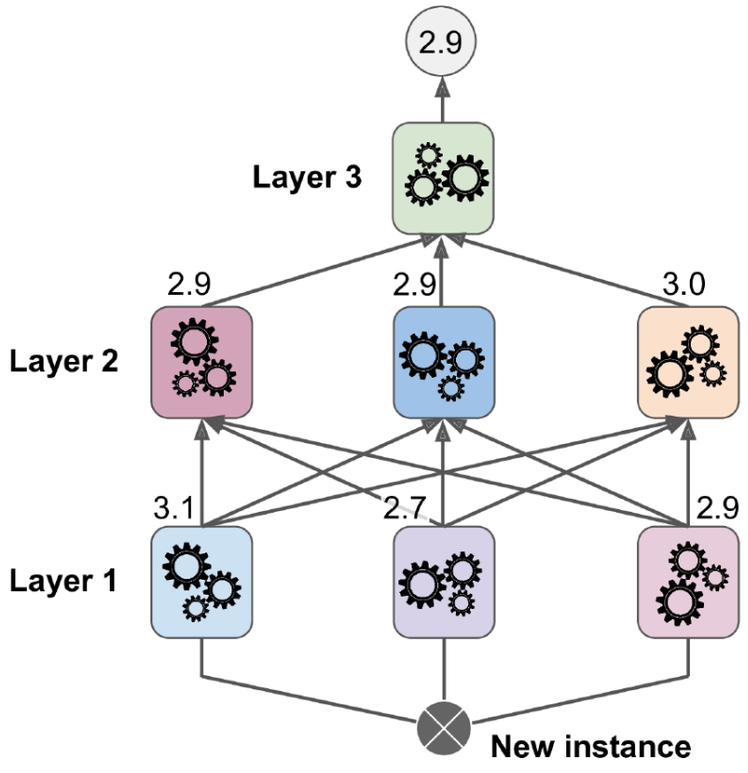
- Un approccio che spesso si impiega per addestrare il blender è il **hold-out set**. Inizialmente il training set è suddiviso in 2. Il primo è usato per addestrare i modelli nel primo layer, mentre il secondo (held-out) è usato per creare le predizioni. Per ogni istanza ci sono 3 predizioni. Tali predizioni costituiscono le features di una istanza in un *nuovo training set*, il cui valore target è quello originale. Il blender è addestrato sul nuovo set.





Multilayer Stacking ensemble

- È possibile considerare più blender, ognuno basato su un modello distinto (es. regressione lineare, random forest, etc), ottenendo un nuovo layer.
- In questo caso si suddivide il training set in 3 parti. La prima usata nel primo layer, come nel caso precedente. La seconda usata dai modelli che combinano le predizioni del primo layer. E la restate parte che combina le predizioni del secondo layer.
- Nota: scikit-learn non supporta lo stacking. Ma ci sono librerie open source, es. https://github.com/viisar/brew https://github.com/Menelau/DESlib



MNIST

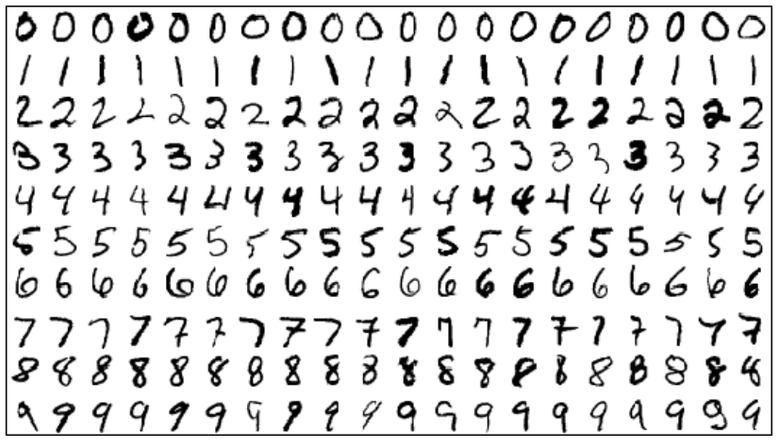
- E’ un dataset molto conosciuto (rielaborato da NIST) di cifre per addestrare sistemi di classificazione basati sulle immagini.
  - "If it doesn't work on MNIST, it won't work at all”; "Well, if it does work on MNIST, it may still fail on others."
- Contiene 60K immagini di addestramento e 10K di training.
  - 1998: un linear classifier ha ottenuto 7.6% di errore rate.
  - 2012: per mezzo di una architettura DL (convolutional neural networks) si è arrivati al 0.23%.
- Ogni immagine è rappresentata in scala di grigi (256 livelli). Le cifre sono centrate in un box 28x28 pixel: abbiamo 784 valori in [0-255] per rappresentare una cifra.
- http://yann.lecun.com/exdb/mnist/
- https://www.kaggle.com/c/digit-recognizer/data
- Implementazione online JS (ott’17) http://myselph.de/neuralNet.html

MNIST: train.csv e test.csv

- Il file train.csv contiene una matrice con 785 colonne. La prima colonna è il *label* della cifra (es. 3) e le restanti colonne sono la rappresentazione sequenziale dell'immagine:

```
000 001 002 003 ... 026 027
028 029 030 031 ... 054 055
056 057 058 059 ... 082 083
 |   |   |   |  ...  |   |
728 729 730 731 ... 754 755
756 757 758 759 ... 782 783
```

- Il file test.csv ha la stessa rappresentazione senza la prima colonna.
- Esempio di immagini:



MNIST: Considerazioni

- Non è impiegato per sistemi avanzati poiché è un task semplice.
  - Algoritmi classici di ML raggiungono i 97% di precisione, approcci Deep Learning il 99.7%
- Troppo utilizzato: si rischia di ideare nuovi approcci adatti solo per questo dataset.
- Molto diverso dai task studiati oggi.

MNIST dataset

- scikit-learn include il dataset che può essere facilmente usato:

```
from sklearn.datasets import fetch_openml

import numpy as np

mnist = fetch_openml('mnist_784', version=1, as_frame=False)
mnist.target = mnist.target.astype(np.uint8)

from sklearn.model_selection import train_test_split

# 50K instanze per il training, 10K validation e 10K test
X_train_val, X_test, y_train_val, y_test = train_test_split(
    mnist.data, mnist.target, test_size=10000, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(
    X_train_val, y_train_val, test_size=10000, random_state=42)
```

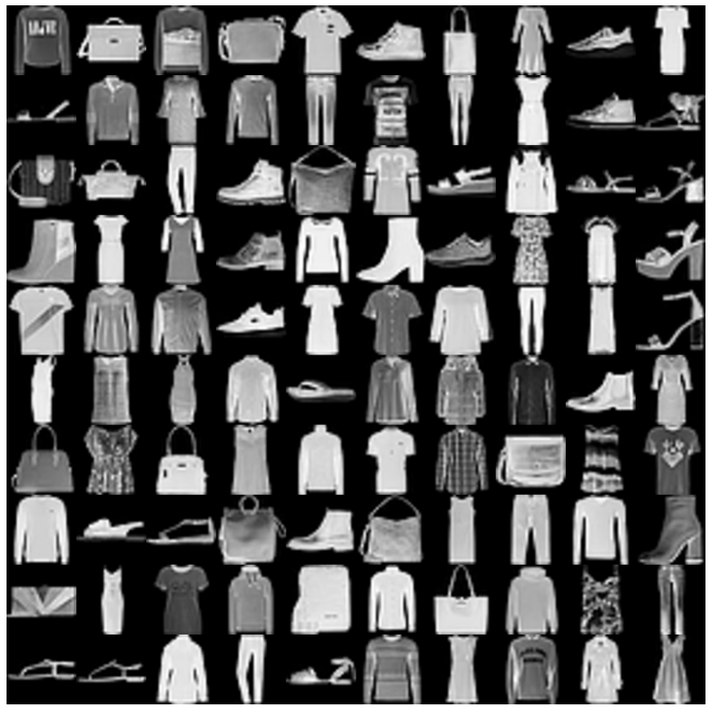
notMNIST

- Simile a MNIST, contiene 10 labels (lettere da A a J), ma ogni lettera nel dataset occorre con font diversi, es:

- http://yaroslavvb.blogspot.fi/2011/09/notmnist-dataset.html
- Download http://yaroslavvb.com/upload/notMNIST/
  - notMNIST_large.tar.gz -> training e validazione
  - notMNIST_small.tar.gz -> test

fashion-MNIST

- Fornito da Zalando. 10 classi che fanno riferimento a generi di vestiario (es. sandali, t-shirt, borse, etc).
- Contiene 60K immagini di addestramento e 10K di training.
- Ogni immagine è rappresentata in scala di grigi di 28x28 pixel



- https://github.com/zalandoresearch/fashion-mnist
- Side-by-side accuracy MNIST vs fashion MNIST:
  - http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/#

Altri dataset popolari sulle immagini

- **CIFAR-10 (e 100)**: 60K 32x32 colour images in 10 classes.
- **ImageNet**: 1,5 milioni di immagini organizzate etichettate su WordNet. In media 1K immagini per concetto.
- **ILSVRC2012 task 1**: 10 milioni di immagini e +1K classi.
- **Open Image**: 9 milioni di URLs di immagini annotate con bounding boxes e migliaia di classi.
- **VisualQA**: open-ended questions su 265K immagini. In media 5.4 questions per immagini con 10 ground truth answers per question.
- **The Street View House Numbers**: 600K immagini di numeri civici.

- Risultati sperimentali ottenuti per varie architetture avanzate:
  - http://rodrigob.github.io/are_we_there_yet/build/#datasets

Esercitazione: Voting Classifier

- Impiega il dataset MNIST con uno split 50K/10K/10K. Scegli almeno tre classificatori e addestrali singolarmente.
- Crea un ensemble Voting, e valutalo sia con approccio soft che hard voting, sia sul validation sia sul test set.
- Confronta i risultati con i classificatori singoli.
- Prova a rimuovere il classificatore che si comporta meglio e valuta nuovamente le prestazioni.

Esercitazione: Stacking Ensemble

- Esegui i singoli classificatori scelti in precedenza e colleziona gli output sul validation set.
- Crea un nuovo training set con tali predizioni. Ogni istanza del set è una vettore che contiene l'insieme di predizioni per una certa immagine, e il target e la classe associata all'immagine. Addestra un classificatore con tale training set. Valutalo sul test set.
- Hai appena realizzato un Stacking ensemble.

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