Lecture 17 - Mar 6

Gradient Boosting
XGBoost

References

- *Elements of Statistical Learning*
  - Ch 10: Boosting and Additive Trees

- T. Chen and C. Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*, 785–794. https://arxiv.org/abs/1603.02754

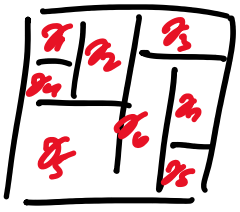
- https://medium.com/syncedreview/tree-boosting-with-xgboost-why-does-xgboost-win-every-machine-learning-competition-ca8034c0b283

Upcoming Deadlines

Homework 4 (Mar 15)

Trees split space into $R_1, R_2, \ldots, R_J$ such that prediction is

$$x \in R_j \Rightarrow f(x) = \gamma_i$$



A tree can be expressed

$$T(x;\Theta) = \sum_{j=1}^{J} \gamma_j \mathbb{1}_{(x \in R_j)}$$

with parameters $\Theta = \{R_j, \gamma_j\}_1^J$ (J: hyperparameter) found by minimizing the empirical risk

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_i)$$

Difficult... so we usually go for suboptimal solutions by

(1) Find γ_j given R_j : easy, $\bar{y}_j$ or model class

(2) Find R_j : hard, approximate. Typical strategy is to use a greedy, recursive partitioning. Sometimes, estimate $\hat{\Theta}$ with a more convenient criterion:

$$\tilde{\Theta} = \arg\min_{\Theta} \sum_{i=1}^{N} \tilde{L}(y_i, T(x_i, \Theta))$$

Then estimate γ_j more precisely.

[illegible] 9.2 for decision tree classifier via Gini

Similar process in 9.2 for decision tree classifier via Gini index instead of loss to identify R_j's.

Boosted tree model

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m)$$

induced by forward-stagewise approach. At each stage, solve

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i, \Theta_m))$$

← Solving is slow, so we need approximations frequently

for the region set and constants $\Theta_m = \{R_{jm}, \gamma_{jm}\}_1^{J_m}$ of the next tree.

Given R_{jm}, finding optimal γ_{jm} is easy

$$\hat{\gamma}_{jm} = \arg\min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm})$$

Two-class classification + exp. loss → AdaBoost

Otherwise,

$e^{-y_i f_{m-1}(x_i)}$

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} w_i^{(m)} \exp(-y_i T(x_i, \Theta_m))$$

$$\hat{\theta}_m = \arg\min_{\theta_m} \sum_{i=1}^{N} w_i^{(m)} \exp\left(-y_i T(x_i, \theta_m)\right)$$

A decision tree algorithm can use exp loss as splitting criterion

$$\Rightarrow \quad \hat{\gamma}_{jm} = \frac{1}{2} \ln\left(\frac{\sum_{x_i \in R_{jm}} w_i^{(m)} \mathbb{1}_{\{y_i = 1\}}}{\sum_{x_i \in R_{jm}} w_i^{(m)} \mathbb{1}_{\{y_i = -1\}}} \right)$$

This requires a specialized tree-growing algorithm, but we prefer weighted least squares regression tree.

Fast approximate algorithms for solving

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L\left(y_i, f_{m-1}(x_i) + T(x_i, \Theta_m)\right)$$

with any differentiable loss can be derived similar to numerical optimization by minimizing

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i))$$

with respect to f of the form $f(x) = \sum_{m=1}^{M} T(x, \Theta_m)$, e.g.

$$\hat{f} = \arg\min_{f} L(f)$$

where the "parameters" $f \in \mathbb{R}^N$ are $f = \{f(x_1), \ldots, f(x_N)\}$.

Numerical optimization solves as

$$f_M = \sum_{m=0}^{M} h_m, \quad h_m \in \mathbb{R}^N,$$

where $f_0 = h_0$ is an initial guess and each f_m is induced based on current vector f_{m-1}, the sum of previous updates.

Different procedures compute "steps" h_m differently.

Different procedures compute steps from ... /

10.10.1 Steepest Descent

$\mathbf{h}_m = -\rho_m \mathbf{g}_m$ where $\rho_m \in \mathbb{R}$, $\mathbf{g}_m \in \mathbb{R}^N$ is the gradient of $L(\mathbf{f})$ evaluated at $\mathbf{f} = \mathbf{f}_{m-1}$

$$g_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i) = f_{m-1}(x_i)}$$

line search

and step length is $\rho_m = \arg\min_{\rho} L(\mathbf{f}_{m-1} - \rho \mathbf{g}_m)$, then

$$\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m \mathbf{g}_m$$

very greedy since $L(\mathbf{f})$ is most rapidly decreasing at $\mathbf{f} = \mathbf{f}_{m-1}$.

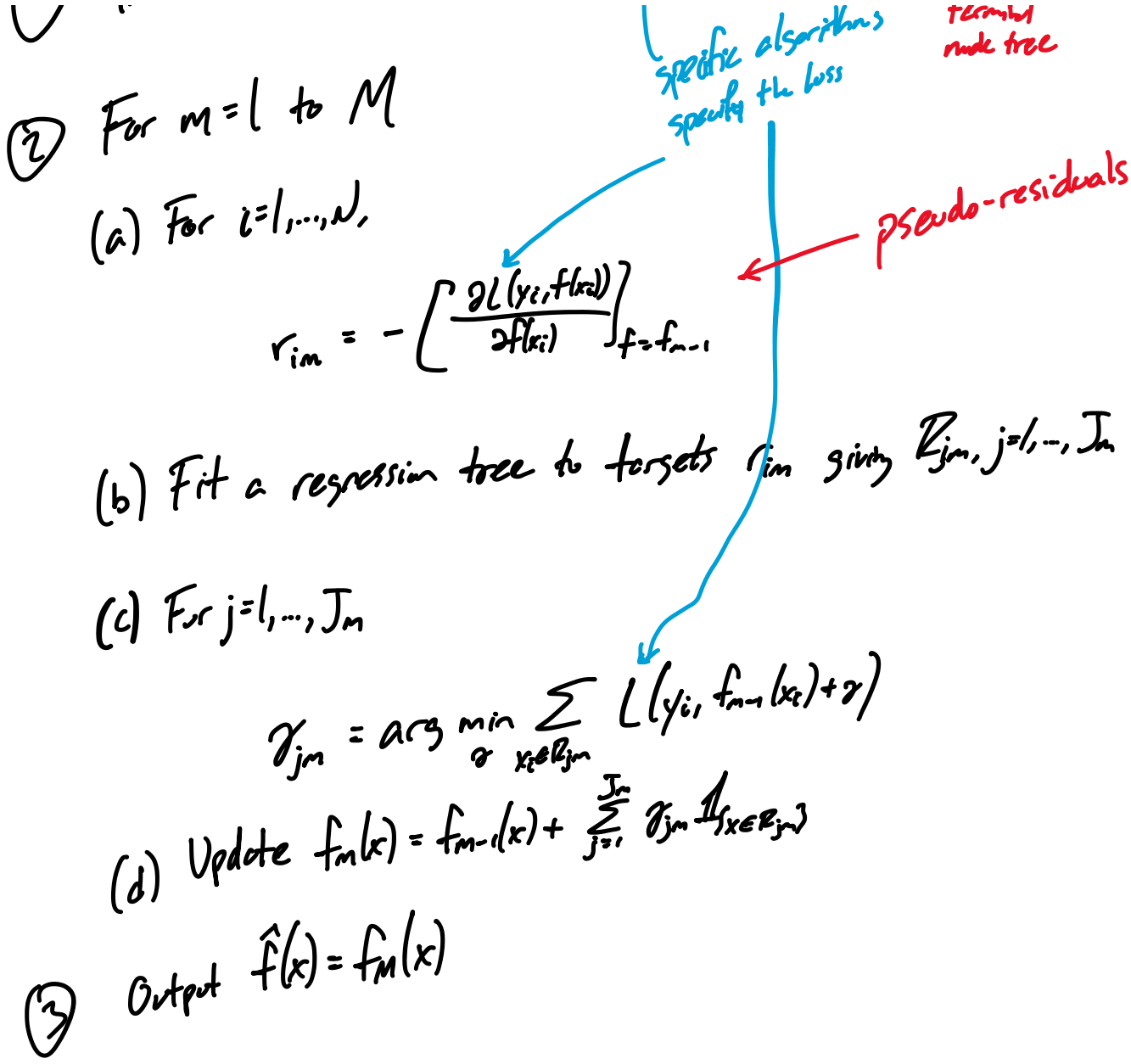
10.10.3 Implementations of Gradient Boosting

generic gradient tree-boosting algorithm for regression

Algorithm 10.3

(1) Initialize $f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$ ← initialize a single terminal node tree

↑ specific algorithms ... loss



② For $m = 1$ to M

(a) For $i = 1, \dots, N$,

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f = f_{m-1}}$$

(b) Fit a regression tree to targets r_{im} giving R_{jm}, $j = 1, \dots, J_m$

(c) For $j = 1, \dots, J_m$

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

(d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbb{1}_{[x \in R_{jm}]}$

③ Output $\hat{f}(x) = f_M(x)$

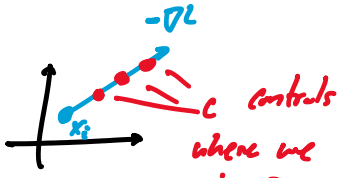
Gradient descent to minimize $L(x)$

$$x_{i+1} = x_i - \alpha \nabla L(x^i)$$

"go in the steepest downhill direction"

where α is a fixed constant

Improving gradient descent



Line search: $x_{i+1} = x_i - c\nabla L(x_i)$

this is how normal gradient boosting works

choose c minimizing $L(x_i - c\nabla L(x_i))$ to pick a distance to jump

easy if f is convex and cheap to compute

If L is infinitely differentiable, the Taylor expansion of L at $x+\Delta x$ is $\sum \frac{a^i}{i!} f^{(i)}(x)$

$$L(x+\alpha\Delta x) = L(x) + \alpha \Delta x^T \nabla L(x) + \frac{\alpha^2}{2!} \|\Delta x\|_2^2 \nabla^2 L(x) + \cdots$$

Gradient descent: $L(x+\alpha\Delta x) \approx L(x) + \alpha \Delta x^T \nabla L(x)$

minimize w.r.t. Δx... we need $\Delta x^T \nabla L(x)$ to be minimized, $\Delta x^T \nabla L(x) \leq 1$

minimize w.r.t. Δx... we need

$$-1 \le \cos\beta = \frac{\Delta x^T \nabla L(x)}{\|\Delta x\| \, \|\nabla L(x)\|} \le 1$$

Δx at angle β to $\nabla L(x)$

$$-\|\Delta x\| \, \|\nabla L(x)\| \le \Delta x^T \nabla L(x)$$

$\beta = \pi$ here, so $\Delta x^T = -\nabla L(x)$ → move opposite direction to gradient

Newton's method:

$$L(x + \Delta x) \approx L(x) + \Delta x^T \nabla L(x) + \frac{1}{2} \|\Delta x\|_2^2 \nabla^2 L(x)$$

minimize right side w.r.t. Δx

$$0 = \frac{\partial}{\partial \Delta x} \left(L(x) + \Delta x^T \nabla L(x) + \frac{1}{2} \|\Delta x\|^2 \nabla^2 L(x) \right)$$

$$0 = \nabla L(x) + \Delta x^T \nabla^2 L(x)$$

$$-\nabla L(x) = \Delta x^T \nabla^2 L(x)$$

$$-\nabla L(x) \left[\nabla^2 L(x)\right]^{-1} = \Delta x^T \nabla L(x) \left[\nabla^2 L(x)\right]^{-1}$$

$$\Rightarrow x_{i+1} = x_i - \nabla L(x) \left[\nabla^2 L(x)\right]^{-1}$$

$Z = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$

Tree ensemble

$$\hat{y}_i = f(x_i) = \sum_{k=1}^{K} T_k(x_i, \theta_k)$$

trees (e.g. CART) with weights w

Let T = number of leaves on tree

Regularized objective

$$L(f) = \sum_{i=1}^{n} L(y_i, f(x_i)) + \sum_{k=1}^{K} \Omega(T_k)$$

differentiable + convex loss function

where $\Omega(T_k) = \gamma T + \frac{1}{2}\lambda \|w_k\|^2$

γ: hyperparam, penalize big trees

λ: hyperparam, L^2 penalty

The model is trained additively, so we have iterations

where we minimize

$$\text{Loss}^t = \sum_{i=1}^{n} L(y_i, f_{t-1}(x_i) + T_t(x_i; \theta_t)) + \Omega(T_k)$$

Taylor expansion at f
(T+f-f = T as product)

Second-order approximation

$$\text{Loss}^t \approx \sum_{i=1}^{n} \left(L(y_i, f_{t-1}(x_i)) + \frac{\partial L(y_i, f_{t-1}(x_i))}{\partial f_{t-1}(x_i)} T_t(x_i; \theta_t) + \frac{1}{2} \frac{\partial^2 L(y_i, f_{t-1}(x_i))}{\partial f_{t-1}(x_i)^2} T_t(x_i; \theta_k)^2 \right)$$

$$+ \Omega(T_t)$$

regularization term

Taylor's theorem... Newton's method

If we do $\arg\min_{\theta_t} \text{loss}^{(t)}$, this is independent of $L(y, f_{t-1}(x_i))$, so

we can solve

$$\underset{\theta_t}{\operatorname{argmin}} \sum_{i=1}^{n} \left[\frac{\partial L(y_i, f_{t-1}(x_i))}{\partial f_{t-1}(x_i)} T_t(x_i; \theta_t) + \frac{1}{2} \frac{\partial^2 L(y_i, f_{t-1}(x_i))}{\partial f_{t-1}(x_i)^2} T_t(x_i; \theta_t)^2 \right] + \Omega(T_t)$$

$$\gamma T + \frac{1}{2} \sum_{j=1}^{T} w_j^2$$

$$\underset{\theta_t}{\operatorname{argmin}} \sum_{j=1}^{T} \left[\left(\sum_{i \in R_j} \frac{\partial L(y_i, f_{t-1}(x_i))}{\partial f_{t-1}(x_i)} \right) w_j + \frac{1}{2} \left(\sum_{i \in R_j} \frac{\partial^2 L(y_i, f_{t-1}(x_i))}{\partial f_{t-1}(x_i)^2} + \lambda \right) w_j^2 \right] + \gamma T$$

g

$$\frac{\partial g}{\partial w_j} = \sum_{j=1}^{T} \left[\left(\sum_{i \in R_j} \frac{\partial L(y_i, f_{t-1}(x_i))}{\partial f_{t-1}(x_i)} \right) + \left(\sum_{i \in R_j} \frac{\partial^2 L(y_i, f_{t-1}(x_i))}{\partial f_{t-1}(x_i)^2} + \lambda \right) w_j \right] = 0$$

$$w_j^* = - \frac{\sum_{i \in R_j} \frac{\partial L(y_i, f_{t-1}(x_i))}{\partial f_{t-1}(x_i)}}{\sum_{i \in R_j} \frac{\partial^2 L(y_i, f_{t-1}(x_i))}{\partial f_{t-1}(x_i)^2} + \lambda}$$

optimal weights

For a fixed tree structure,

$$\Rightarrow \text{loss}^t = - \sum_{j=1}^{T} \frac{\left(\sum_{i \in R_j} \frac{\partial L(y_i, f_{t-1}(x_i))}{\partial f_{t-1}(x_i)} \right)^2}{\sum_{i \in R_j} \frac{\partial^2 L(y_i, f_{t-1}(x_i))}{\partial f_{t-1}(x_i)^2} + \lambda} + \frac{1}{2} \left(\sum_{i \in R_j} \frac{\partial^2 L(y_i, f_{t-1}(x_i))}{\partial f_{t-1}(x_i)^2} + \lambda \right) \left(- \frac{\left(\sum_{i \in R_j} \frac{\partial L}{\partial f_{t-1}} \right)}{\sum_{i \in R_j} \frac{\partial^2 L}{\partial f_{t-1}^2} + \lambda} \right)^2 + \gamma T$$

$$\text{loss}^m = \frac{-1}{2} \sum_{j=1}^{|T_m|} \frac{\left(\sum_{i \in R_j} \frac{\partial L}{\partial f_{t-1}} \right)^2}{\left(\sum_{i \in R_j} \frac{\partial^2 L}{\partial f_{t-1}^2} + \lambda \right)} + \gamma |T_m|$$

scoring function for quality of a tree structure T_t

like impurity but derived from a wider class of objective functions

a tree structure T_t

$$Loss^{split} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L}\frac{\partial l}{\partial f_{t-1}}\right)^2}{\sum_{i\in I_L}\frac{\partial^2 l}{\partial f_{t-1}^2}+\lambda} + \frac{\left(\sum_{i\in I_R}\frac{\partial l}{\partial f_{t-1}}\right)^2}{\sum_{i\in I_R}\frac{\partial^2 l}{\partial f_{t-1}}+\lambda} - \frac{\left(\sum_{i\in I}\frac{\partial l}{\partial f_{t-1}}\right)^2}{\sum_{i\in I}\frac{\partial^2 l}{\partial f_{t-1}^2}+\lambda}\right] - \gamma$$

for evaluating split candidates

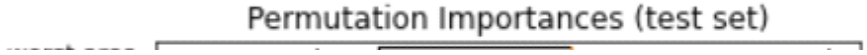
Example

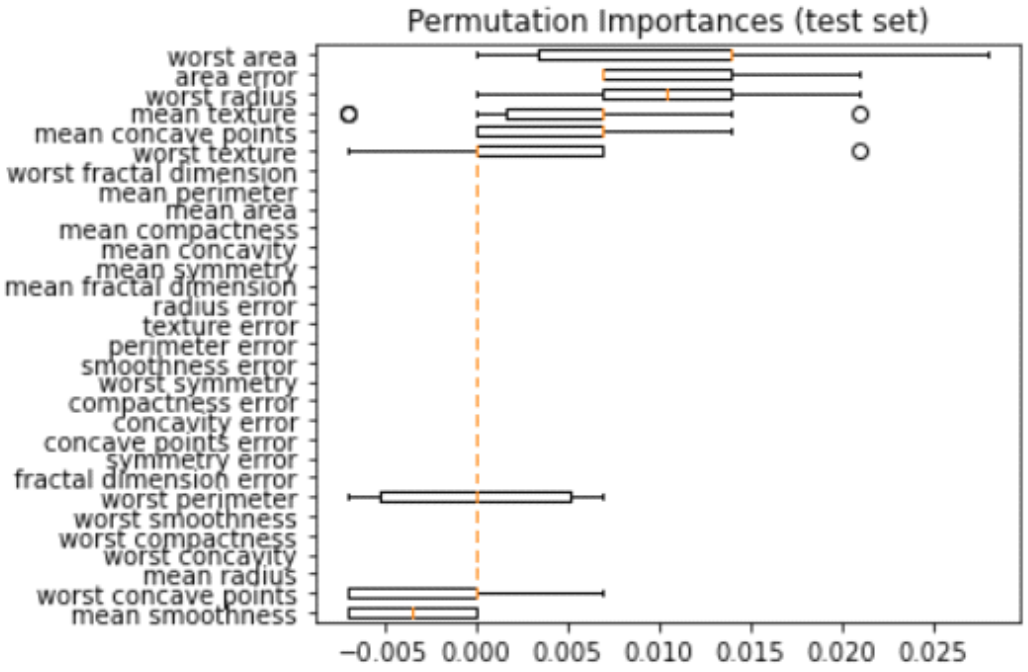
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn

from sklearn.datasets import load_breast_cancer
from sklearn.datasets import load_wine
from sklearn.inspection import permutation_importance
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split

from tensorflow.keras.datasets import mnist
from tensorflow.keras.datasets import cifar10

from xgboost import XGBClassifier
```

Breast Cancer Classification

```
cancer = load_breast_cancer()

X = cancer['data']
Y = cancer['target']

# split the data into train and test sets
trainX, testX, trainY, testY = train_test_split(X, Y, test_size = 0.25)

# build the classifier
model = XGBClassifier(n_jobs = -1)
```

```
result = permutation_importance(model, testX, testY, n_repeats = 10, n_jobs = -1)
sorted_idx = result.importances_mean.argsort()

fig, ax = plt.subplots()
ax.boxplot(result.importances[sorted_idx].T, vert = False,
           labels = cancer.feature_names[sorted_idx])
ax.set_title("Permutation Importances (test set)")
fig.tight_layout()
plt.show()
```




Permutation Importances (test set)
worst area
area error
worst radius
mean texture
mean concave points
worst texture
worst fractal dimension
mean perimeter
mean area
mean compactness
mean concavity
mean symmetry
mean fractal dimension
radius error
texture error
perimeter error
smoothness error
worst symmetry
compactness error
concavity error
concave points error
symmetry error
fractal dimension error
worst perimeter
worst smoothness
worst compactness
worst concavity
mean radius
worst concave points
mean smoothness
−0.005
0.000
0.005
0.010
0.015
0.020
0.025

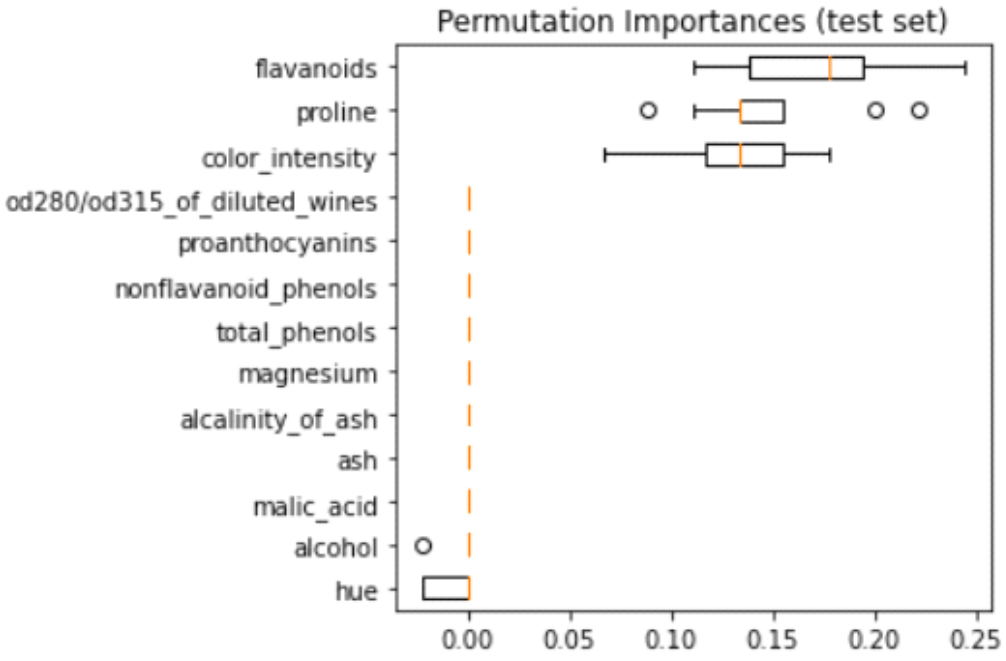
Example

Wine Example

```
wine = load_wine()

X = wine['data']
Y = wine['target']

# split the data into train and test sets
trainX, testX, trainY, testY = train_test_split(X, Y, test_size = 0.25)

# build the classifier
model = XGBClassifier(n_jobs = -1)
```

...

```
result = permutation_importance(model, testX, testY, n_repeats = 10, n_jobs = -1)
sorted_idx = result.importances_mean.argsort()

fig, ax = plt.subplots()
ax.boxplot(result.importances[sorted_idx].T, vert = False,
           labels = np.array(wine.feature_names)[sorted_idx])
ax.set_title("Permutation Importances (test set)")
fig.tight_layout()
plt.show()
```



Example

MNIST Example

```
(trainX, trainY), (testX, testY) = mnist.load_data()

# preprocess the data
trainX = trainX.reshape(trainX.shape[0], trainX.shape[1] * trainX.shape[2])
trainX = trainX.astype('float')/255.0

testX = testX.reshape(testX.shape[0], testX.shape[1] * testX.shape[2])
testX = testX.astype('float')/255.0

# build the classifier
model = XGBClassifier(n_jobs = -1)
```

… 100% train, 98% test

Let's try to tune XGBoost with 5-fold cross-validation to see if we can improve the results.

```
parameters = {'max_depth': [1, 5, 10, 100, None], 'reg_lambda': [0.001, 0.01, 0.1, 1, 10]}
model = GridSearchCV(XGBClassifier(n_jobs = -1, eval_metric='mlogloss'), parameters, n_jobs = -1)

# fit the classifier to the training data
model.fit(trainX, trainY)
```

No improvements…

Example

CIFAR10 Example

```
(trainX, trainY), (testX, testY) = cifar10.load_data()

# preprocess the data
trainX = trainX.reshape(trainX.shape[0], trainX.shape[1] * trainX.shape[2] * 3)
trainX = trainX.astype('float')/255.0

testX = testX.reshape(testX.shape[0], testX.shape[1] * testX.shape[2] * 3)
testX = testX.astype('float')/255.0

# build the classifier
model = XGBClassifier(n_jobs = -1)
```

Training Classification Report:

	precision	recall	f1-score	support
0	0.98	0.99	0.98	5000
1	1.00	1.00	1.00	5000
2	0.97	0.98	0.97	5000
3	0.99	0.97	0.98	5000
4	0.96	0.98	0.97	5000
5	0.99	0.98	0.99	5000
6	0.98	0.99	0.99	5000
7	1.00	0.99	0.99	5000
8	0.99	0.99	0.99	5000
9	1.00	0.99	1.00	5000
accuracy			0.99	50000
macro avg	0.99	0.99	0.99	50000
weighted avg	0.99	0.99	0.99	50000

Testing Classification Report:

	precision	recall	f1-score	support
0	0.61	0.62	0.62	1000

0	0.61	0.62	0.62	1000
1	0.64	0.63	0.64	1000
2	0.43	0.39	0.41	1000
3	0.39	0.38	0.39	1000
4	0.48	0.45	0.47	1000
5	0.48	0.48	0.48	1000
6	0.54	0.63	0.58	1000
7	0.60	0.57	0.59	1000
8	0.64	0.67	0.66	1000
9	0.58	0.60	0.59	1000
accuracy			0.54	10000