CSI5180. Machine Learning for Bioinformatics Applications

Ensemble Learning

by

Marcel Turcotte

Version **December 5, 2019**

Preamble

Preamble

Ensemble Learning

In this lecture, we consider several meta learning algorithms all based on the principle that the combined opinion of a large group of individuals is often more accurate than the opinion of a single expert — this is often referred to as the **wisdom of the crowd**. Today, we tell apart the following meta-algorithms: **bagging**, **pasting**, **random patches**, **random subspaces**, **boosting**, and **stacking**.

General objective :

- **Compare** the specific features of various ensemble learning meta-algorithms

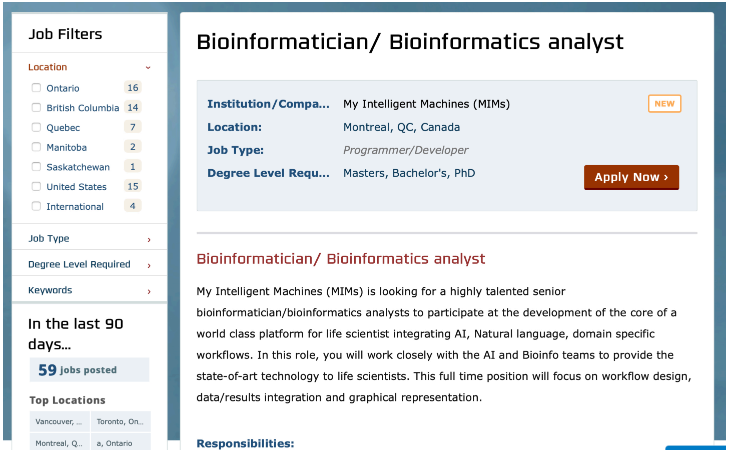
Learning objectives

- **Discuss** the intuition behind bagging and pasting methods
- **Explain** the difference between random patches and random subspaces
- **Describe** boosting methods
- **Contrast** the stacking meta-algorithms from bagging

Reading:

- Jaswinder Singh, Jack Hanson, Kuldip Paliwal, and Yaoqi Zhou. RNA secondary structure prediction using an ensemble of two-dimensional deep neural networks and transfer learning. *Nature Communications* **10**(1):5407, 2019.

www.mims.ai



Job Filters

Location

Ontario	16
British Columbia	14
Quebec	7
Manitoba	2
Saskatchewan	1
United States	15
International	4

Job Type

Degree Level Required

Keywords

In the last 90 days...

59 jobs posted

Top Locations

Vancouver, ... Toronto, On... Montreal, Q... a, Ontario

Bioinformatician/ Bioinformatics analyst

Institution/Compa...	My Intelligent Machines (MIMs)
Location:	Montreal, QC, Canada
Job Type:	*Programmer/Developer*
Degree Level Requ...	Masters, Bachelor's, PhD

Bioinformatician/ Bioinformatics analyst

My Intelligent Machines (MIMs) is looking for a highly talented senior bioinformatician/bioinformatics analysts to participate at the development of the core of a world class platform for life scientist integrating AI, Natural language, domain specific workflows. In this role, you will work closely with the AI and Bioinfo teams to provide the state-of-art technology to life scientists. This full time position will focus on workflow design, data/results integration and graphical representation.

Responsibilities:

bioinformatics.ca/job-postings

Plan

Introduction

Ensemble Learning - What is it?

- "**Ensemble learning** is a learning paradigm that, instead of trying to learn one super-accurate model, focuses on **training a large number of low-accuracy models** and then **combining the predictions** given by those weak models to obtain a **high-accuracy meta-model**." [Burkov, 2019] §7.5

Ensemble Learning - What is it?

- "**Ensemble learning** is a learning paradigm that, instead of trying to learn one super-accurate model, focuses on **training a large number of low-accuracy models** and then **combining the predictions** given by those weak models to obtain a **high-accuracy meta-model**." [Burkov, 2019] §7.5
- **Weak learners** (low-accuracy) models are simple and fast, both for training and prediction.

Ensemble Learning - What is it?

- "**Ensemble learning** is a learning paradigm that, instead of trying to learn one super-accurate model, focuses on **training a large number of low-accuracy models** and then **combining the predictions** given by those weak models to obtain a **high-accuracy meta-model**." [Burkov, 2019] §7.5
- **Weak learners** (low-accuracy) models are simple and fast, both for training and prediction.
- The **general idea** is that **each learner has a vote**, and these votes are **combined** to establish the final decision.

Ensemble Learning - What is it?

- "**Ensemble learning** is a learning paradigm that, instead of trying to learn one super-accurate model, focuses on **training a large number of low-accuracy models** and then **combining the predictions** given by those weak models to obtain a **high-accuracy meta-model**." [Burkov, 2019] §7.5
- **Weak learners** (low-accuracy) models are simple and fast, both for training and prediction.
- The **general idea** is that **each learner has a vote**, and these votes are **combined** to establish the final decision.
- **Decision trees** are the most commonly used weak learners.

Ensemble Learning - What is it?

- "**Ensemble learning** is a learning paradigm that, instead of trying to learn one super-accurate model, focuses on **training a large number of low-accuracy models** and then **combining the predictions** given by those weak models to obtain a **high-accuracy meta-model**." [Burkov, 2019] §7.5
- **Weak learners** (low-accuracy) models are simple and fast, both for training and prediction.
- The **general idea** is that **each learner has a vote**, and these votes are **combined** to establish the final decision.
- **Decision trees** are the most commonly used weak learners.
- Ensemble learning is fact an umbrella for a large family of meta-algorithms, including **bagging**, **pasting**, **random patches**, **random subspaces**, **boosting**, and **stacking**.

Justification

Weak learners/high accuracy

- 10 experiments

See: [Géron, 2019] §7

Weak learners/high accuracy

- 10 experiments
  - Each experiment consists of tossing a loaded coin

See: [Géron, 2019] §7

Weak learners/high accuracy

- 10 experiments
  - Each experiment consists of tossing a loaded coin
    - 51 % head, 49 % tail

See: [Géron, 2019] §7

Weak learners/high accuracy

- 10 experiments
    - Each experiment consists of tossing a loaded coin
        - 51 % head, 49 % tail
    - As the number of toss increases, the proportion of heads will approach 51%

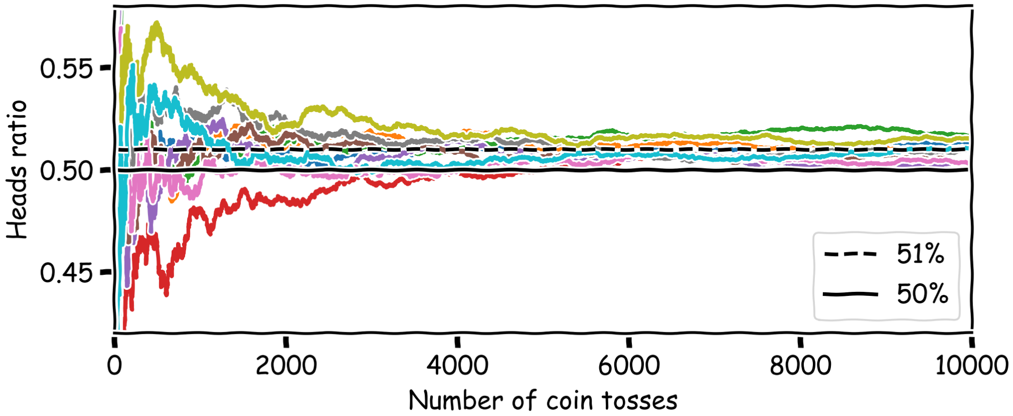
See: [Géron, 2019] §7

Source code

```python
tosses = (np.random.rand(10000, 10) < 0.51).astype(np.int8)
cumsum = np.cumsum(tosses, axis=0) / np.arange(1, 10001).reshape(-1, 1)

with plt.xkcd():
    plt.figure(figsize=(8,3.5))
    plt.plot(cumsum)
    plt.plot([0, 10000], [0.51, 0.51], "k--", linewidth=2, label="51%")
    plt.plot([0, 10000], [0.5, 0.5], "k-", label="50%")
    plt.xlabel("Number of coin tosses")
    plt.ylabel("Heads ratio")
    plt.legend(loc="lower right")
    plt.axis([0, 10000, 0.42, 0.58])
    plt.tight_layout()
    plt.savefig("weak_learner.pdf", format="pdf", dpi=264)
```

See: [Géron, 2019] §7

Weak learners/high accuracy



Adapted from [Géron, 2019] §7

Independent learners

- Clearly, the **learners** are using the same input, they are **not independent**.

Independent learners

- Clearly, the **learners** are using the same input, they are **not independent**.
- **Ensemble learning** works best when the learners are as **independent** one from another as possible.

Independent learners

- Clearly, the **learners** are using the same input, they are **not independent**.
- **Ensemble learning** works best when the learners are as **independent** one from another as possible.
  - Different **algorithms**

Independent learners

- Clearly, the **learners** are using the same input, they are **not independent**.
- **Ensemble learning** works best when the learners are as **independent** one from another as possible.
  - Different **algorithms**
  - Different **sets of features**

Independent learners

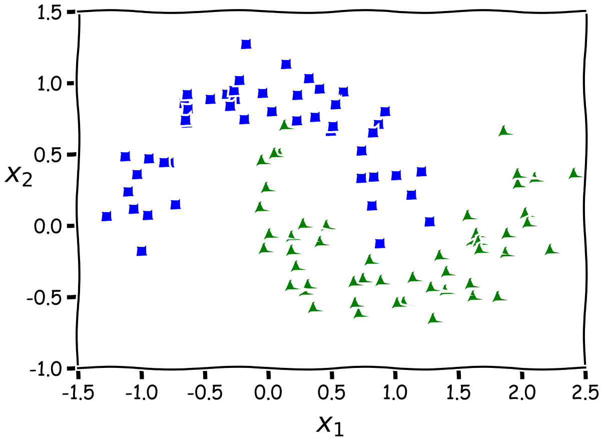
- Clearly, the **learners** are using the same input, they are **not independent**.
- **Ensemble learning** works best when the learners are as **independent** one from another as possible.
  - Different **algorithms**
  - Different **sets of features**
  - Different **data sets**

Data set - moons

```python
import matplotlib.pyplot as plt

from sklearn.datasets import make_moons

X, y = make_moons(n_samples=100, noise=0.15)

with plt.xkcd():
    plt.plot(X[:, 0][y==0], X[:, 1][y==0], "bs")
    plt.plot(X[:, 0][y==1], X[:, 1][y==1], "g^")
    plt.axis([-1.5, 2.5, -1, 1.5])
    plt.grid(True, which='both')
    plt.xlabel(r"$x_1$", fontsize=20)
    plt.ylabel(r"$x_2$", fontsize=20, rotation=0)
    plt.tight_layout()
    plt.savefig("make_moons.pdf", format="pdf", dpi=264)
```

Adapted from: [Géron, 2019] §5

Data set - moons



Adapted from [Géron, 2019] §5

Source code - VotingClassifier - hard

```
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

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

estimators=[('lr', log_clf),
            ('rf', rnd_clf),
            ('svc', svm_clf)]

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

voting_clf.fit(X_train, y_train)
```

Source: [Géron, 2019] §7

Source code - accuracy

```
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))
```

Source code - accuracy

```python
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
```

Source code - accuracy

```python
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
```

[Géron, 2019] §7

Source code - VotingClassifier - soft

```
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC(probability=True)

estimators=[('lr', log_clf),
            ('rf', rnd_clf),
            ('svc', svm_clf)]

voting_clf = VotingClassifier(estimators=estimators, voting='soft')

voting_clf.fit(X_train, y_train)
```

Source: [Géron, 2019] §7

Source code - accuracy

```
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))
```

Source code - accuracy

```python
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
```

Source code - accuracy

```python
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
```

- **Soft** uses the average probability score, rather than hard voting.

[Géron, 2019] §7

Meta-algorithms

Bagging and pasting

- **Ensemble learning** works best when the learners are independent.

Bagging and pasting

- **Ensemble learning** works best when the learners are independent.
- One way to achieve this is to train the learners on **(slightly) different data sets**.

Bagging and pasting

- **Ensemble learning** works best when the learners are independent.
- One way to achieve this is to train the learners on **(slightly) different data sets**.
    - **Bagging**: sampling **with replacement** (bootstrap aggregating);

Bagging and pasting

- **Ensemble learning** works best when the learners are independent.
- One way to achieve this is to train the learners on **(slightly) different data sets**.
  - **Bagging**: sampling **with replacement** (bootstrap aggregating);
  - **Pasting**: sampling **without replacement**.

Bagging and pasting

- **Ensemble learning** works best when the learners are independent.
- One way to achieve this is to train the learners on **(slightly) different data sets**.
  - **Bagging**: sampling **with replacement** (bootstrap aggregating);
  - **Pasting**: sampling **without replacement**.
- As an added bonus, the learns can be trained in **parallel**!

Bagging and pasting

- **Ensemble learning** works best when the learners are independent.
- One way to achieve this is to train the learners on **(slightly) different data sets**.
  - **Bagging**: sampling **with replacement** (bootstrap aggregating);
  - **Pasting**: sampling **without replacement**.
- As an added bonus, the learns can be trained in **parallel**!
- Literature suggests that **bagging** outperforms **pasting** [Géron, 2019].

sklearn.ensemble.BaggingClassifier

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
            DecisionTreeClassifier(),
            n_estimators=500, max_samples=100,
            bootstrap=True, n_jobs=8
          )

bag_clf.fit(nX_train, y_train)

y_pred = bag_clf.predict(X_test)
```

- **Soft voting** by default
- **bootstrap=False** implies **pasting**

Adapted from: [Géron, 2019] §7

Not just for classification

- **Bagging** and **pasting** apply for **regression** tasks as well.
    - **BaggingRegressor** in **Keras**
    - **Voting** is replaced the **average**

Claim

- **Claim:**

Claim

- **Claim:**
    - On **average 37 %** of the training examples are **not used** when **bagging**!

Claim

- **Claim:**
  - On **average 37 %** of the training examples are **not used** when **bagging**!
- By default, bagging samples N examples **with replacement**, where N is the size of the training set.

Empirical evidence

```
from random import random

def do_sample_with_replacement():

    xs = [ 1 for i in range(100) ]

    for sample in range(100):
        index = int(100 * random())
        xs[ index ] = 0

    print(sum(xs))

for run in range(10):
    do_sample_with_replacement()
```

Empirical evidence

```
38
33
34
37
37
37
44
37
35
37
```

Out-of-bag evaluation (oob)

Out-of-bag evaluation (oob)

- By default, bagging samples N examples **with replacement**, where N is the size of the training set.

Out-of-bag evaluation (oob)

- By default, bagging samples N examples **with replacement**, where N is the size of the training set.
- This means that **on average**, for each learner, 37% of the examples are not used.

Out-of-bag evaluation (oob)

- By default, bagging samples N examples **with replacement**, where N is the size of the training set.
- This means that **on average**, for each learner, 37% of the examples are not used.
- These **unseen**, out-of-bag, examples can be used for validation!

Out-of-bag evaluation (oob)

- By default, bagging samples N examples **with replacement**, where N is the size of the training set.
- This means that **on average**, for each learner, 37% of the examples are not used.
- These **unseen**, out-of-bag, examples can be used for validation!
- OOB (possibly) **eliminates the need** for a separate **validation set**.

Out-of-bag evaluation (oob)

- By default, bagging samples N examples **with replacement**, where N is the size of the training set.
- This means that **on average**, for each learner, 37% of the examples are not used.
- These **unseen**, out-of-bag, examples can be used for validation!
- OOB (possibly) **eliminates the need** for a separate **validation set**.

Out-of-bag evaluation (oob)

- By default, bagging samples N examples **with replacement**, where N is the size of the training set.
- This means that **on average**, for each learner, 37% of the examples are not used.
- These **unseen**, out-of-bag, examples can be used for validation!
- OOB (possibly) **eliminates the need** for a separate **validation set**.

```
bag_clf = BaggingClassifier(
                DecisionTreeClassifier(), n_estimators=500,
                bootstrap=True, n_jobs=-1, oob_score=True)

bag_clf.fit(X_train, y_train)
print(bag_clf.oob_score_)
```

```
0.9013333333333332
```

Random patches and subspaces

- **BaggingClassifier** also supports **sampling features**.

Random patches and subspaces

- **BaggingClassifier** also supports **sampling features**.
  - This is controlled by the parameters **bootstrap_features** and **max_features**.

Random patches and subspaces

- **BaggingClassifier** also supports **sampling features**.
  - This is controlled by the parameters **bootstrap_features** and **max_features**.
    - **Random patches:** sampling **both** instances and features.

```
bag_clf = BaggingClassifier(
                DecisionTreeClassifier(), n_estimators=500,
                bootstrap=True, max_samples=1.0,
                bootstrap_features=True, max_features=0.4,
                n_jobs=-1, oob_score=True)
```

Random patches and subspaces

- **BaggingClassifier** also supports **sampling features**.
  - This is controlled by the parameters **bootstrap_features** and **max_features**.
    - **Random patches:** sampling **both** instances and features.

```
bag_clf = BaggingClassifier(
            DecisionTreeClassifier(), n_estimators=500,
            bootstrap=True, max_samples=1.0,
            bootstrap_features=True, max_features=0.4,
            n_jobs=-1, oob_score=True)
```

    - **Random subspaces:** **only** sampling features.

Random Forest

```
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(splitter="random", max_leaf_nodes=16),
    n_estimators=500, max_samples=1.0, bootstrap=True)
```

sklearn.ensemble.RandomForestClassifier

- "The **Random Forest** algorithm introduces **extra randomness** when growing trees; instead of searching for the very best feature when splitting a node (. . .), **it searches for the best feature among a random subset of features**." [Géron, 2019]

```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16)

rfc.fit(X_train, y_train)

y_pred_rf = rfc.predict(X_test)
```

- See also **ExtraTreesClassifier** and **ExtraTreesRegressor**.

Boosting

- **Boosting** meta-algorithms are training learners **sequentially**, in such a way that each classifier is trying to **correct the mistakes** of the previous classifier in the chain.

AdaBoost

- **AdaBoost** stands for **Adaptive Boosting**.

AdaBoost

- **AdaBoost** stands for **Adaptive Boosting**.
- Each learner focuses on examples that were **incorrectly classified by the previous classifier**.

AdaBoost

- **AdaBoost** stands for **Adaptive Boosting**.
- Each learner focuses on examples that were **incorrectly classified by the previous classifier**.
    - Specifically, the **weight** of examples **incorrectly** is increased with each iteration.

AdaBoost

- **AdaBoost** stands for **Adaptive Boosting**.
- Each learner focuses on examples that were **incorrectly classified by the previous classifier**.
  - Specifically, the **weight** of examples **incorrectly** is increased with each iteration.
  - Initially, the weight of each example (w_i) is $\frac{1}{N}$, where N is the number of examples.

AdaBoost - error rate

- Let's define an **indicator function**:

$$I(\hat{y}_i^{(j)}, y_i) = \begin{cases} 0 & \text{if } \hat{y}_i^{(j)} = y_i \\ 1 & \text{if } \hat{y}_i^{(j)} \neq y_i \end{cases}$$

where $\hat{y}_i^{(j)}$ is the prediction of the j^{th} learner on example i and y_i is the label of example i.

AdaBoost - error rate

- Let's define an **indicator function**:

$$I(\hat{y}_i^{(j)}, y_i) = \begin{cases} 0 & \text{if } \hat{y}_i^{(j)} = y_i \\ 1 & \text{if } \hat{y}_i^{(j)} \neq y_i \end{cases}$$

  where $\hat{y}_i^{(j)}$ is the prediction of the j^{th} learner on example i and y_i is the label of example i.

- The **error rate** of the j^{th} learner is defined as:

$$r_j = \frac{\sum_{i=1}^{N} w_i \times I(\hat{y}_i^{(j)}, y_i)}{\sum_{i=1}^{N} w_i}$$

AdaBoost - learner's weight

When making a **final decision (vote)**, each learner has a weigth.

- The **weight** of the learner j:

$$\alpha_j = \eta \log \frac{1 - r_j}{r_j}$$

  where η is the learning rate, default value is 1.
  - Low error rate implies high learn's weight.
  - Random guesses, error rate $= 0.5$, implies a weight of 0.
  - Error rate > 0.5 implies a negative weight.

AdaBoost - update

- After training the learner j, the weight of each example is updated as follows.

$$w_i = \begin{cases} w_i & \text{if } \hat{y}_i^{(j)} = y_i \\ w_i \times e^{\alpha_j} & \text{if } \hat{y}_i^{(j)} \neq y_i \end{cases}$$

AdaBoost - update

- After training the learner j, the weight of each example is updated as follows.

$$w_i = \begin{cases} w_i & \text{if } \hat{y}_i^{(j)} = y_i \\ w_i \times e^{\alpha_j} & \text{if } \hat{y}_i^{(j)} \neq y_i \end{cases}$$

- The weights are then normalized, dividing them by $\sum_{i=1}^{N} w_i$

AdaBoost - prediction

- The outcome is the class with the **largest weighted vote**:

$$\hat{y}(x) = \operatorname{argmax}_k \sum_{\substack{j=1 \\ \hat{y}^{(j)}(x)=k}}^{m} \alpha_j$$

  where m is the number of learners.

sklearn.ensemble.AdaBoostClassifier

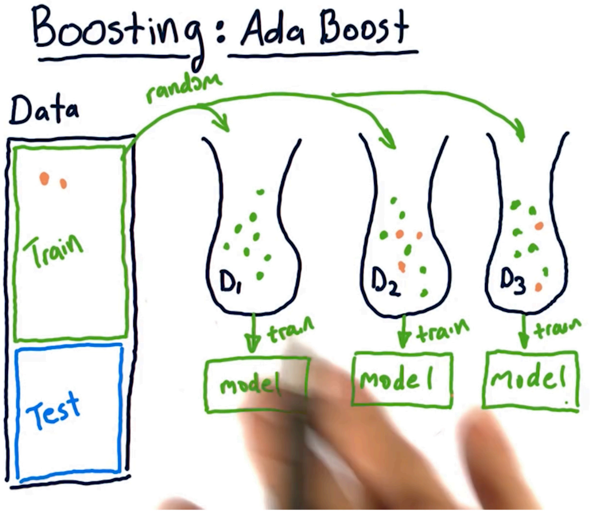
```
from sklearn.ensemble import AdaBoostClassifier

ada_clf = AdaBoostClassifier(
            DecisionTreeClassifier(max_depth=1),
            n_estimators=200,
            algorithm="SAMME.R",
            learning_rate=0.5)

ada_clf.fit(X_train, y_train)
```

[Géron, 2019] §7

AdaBoost

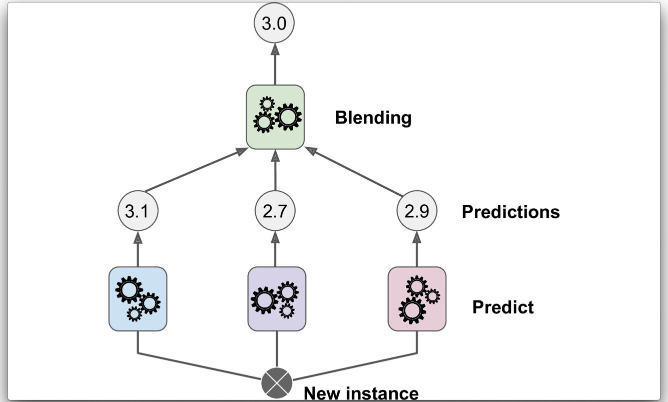
- A literature search using Scopus for “**AdaBoost**” and “**bioinformatics** returns **78 references**. Including the following two papers:
  - Y. Qu, B.-L. Adam, Y. Yasui, M.D. Ward, L.H. Cazares, P.F. Schellhammer, Z. Feng, O.J. Semmes, and G.L. Wright Jr., Boosted decision tree analysis of surface-enhanced laser desorption/ionization mass spectral serum profiles discriminates prostate cancer from noncancer patients, *Clinical Chemistry* **48** (2002), no. 10, 18351843, **cited By 382**.
  - P.M. Long and V.B. Vega, Boosting and microarray data, *Machine Learning* **52** (2003), no. 1-2, 3144, **cited By 40**.

AdaBoost



https://youtu.be/GM3CDQfQ4sw

Stacking



Source [Géron, 2019] Figure 7.12

Stacking

- **Like** bagging, **stacking** combines the predictions of several learners.

Stacking

- **Like** bagging, **stacking** combines the predictions of several learners.
- **Unlike** bagging, **stacking** does not use a predetermined function to combine the predictions, say majority vote, instead, it **trains a classifier/regressor**.

Stacking

- **Like** bagging, **stacking** combines the predictions of several learners.
- **Unlike** bagging, **stacking** does not use a predetermined function to combine the predictions, say majority vote, instead, it **trains a classifier/regressor**.
- A **holdout set** is used to train the **blender**.

Prologue

Summary

- **Ensemble learning** is the idea of combining the predictions of several weak learners.

Summary

- **Ensemble learning** is the idea of combining the predictions of several weak learners.
- **Ensemble learning** works best when the learners are as independent one from another as possible.

Summary

- **Ensemble learning** is the idea of combining the predictions of several weak learners.
- **Ensemble learning** works best when the learners are as independent one from another as possible.
- This diversity of learners can be achieved in various ways: **different algorithms**, **different sets of features**, **(slightly) different data sets**.

Summary

- **Ensemble learning** is the idea of combining the predictions of several weak learners.
- **Ensemble learning** works best when the learners are as independent one from another as possible.
- This diversity of learners can be achieved in various ways: **different algorithms**, **different sets of features**, **(slightly) different data sets**.
- **Boosting** combines the learners in a sequential, rather than parallel, manner. Each learner fixes the mistakes of its predecessor.

Summary

- **Ensemble learning** is the idea of combining the predictions of several weak learners.
- **Ensemble learning** works best when the learners are as independent one from another as possible.
- This diversity of learners can be achieved in various ways: **different algorithms**, **different sets of features**, **(slightly) different data sets**.
- **Boosting** combines the learners in a sequential, rather than parallel, manner. Each learner fixes the mistakes of its predecessor.
- With stacking, a **learning algorithm** is used to combine the results the **weak classifiers**.

Next module

- **Null**

References

Burkov, A. (2019).
The Hundred-Page Machine Learning Book.
Andriy Burkov.

Cao, Z., Pan, X., Yang, Y., Huang, Y., and Shen, H.-B. (2018).
The lncLocator: a subcellular localization predictor for long non-coding rnas based on a stacked ensemble classifier.
Bioinformatics, 34(13):2185–2194.

Chen, X., Zhu, C.-C., and Yin, J. (2019).
Ensemble of decision tree reveals potential miRNA-disease associations.
PLoS Comput Biol, 15(7):e1007209.

Colomé-Tatché, M. and Theis, F. J. (2018).
Statistical single cell multi-omics integration.
Current Opinion in Systems Biology, 7:54–59.

Géron, A. (2019).
Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow.
O'Reilly Media, 2nd edition.

References

Ma, Y., Liu, Y., and Cheng, J. (2018).
Protein secondary structure prediction based on data partition and semi-random subspace method.
Sci Rep, 8(1):9856.

Meher, P. K., Sahu, T. K., Gahoi, S., Satpathy, S., and Rao, A. R. (2019).
Evaluating the performance of sequence encoding schemes and machine learning methods for splice sites recognition.
Gene, 705:113–126.

Peng, H., Zheng, Y., Zhao, Z., Liu, T., and Li, J. (2018).
Recognition of CRISPR/Cas9 off-target sites through ensemble learning of uneven mismatch distributions.
Bioinformatics, 34(17):i757–i765.

Singh, A. P., Mishra, S., and Jabin, S. (2018a).
Sequence based prediction of enhancer regions from DNA random walk.
Sci Rep, 8(1):15912.

References

Singh, J., Hanson, J., Heffernan, R., Paliwal, K., Yang, Y., and Zhou, Y. (2018b).
Detecting proline and non-proline cis isomers in protein structures from sequences using deep residual ensemble learning.
J Chem Inf Model, 58(9):2033–2042.

Singh, J., Hanson, J., Paliwal, K., and Zhou, Y. (2019).
RNA secondary structure prediction using an ensemble of two-dimensional deep neural networks and transfer learning.
Nature Communications, 10(1):5407.

Su, W., Gu, X., and Peterson, T. (2019).
TIR-Learner, a new ensemble method for TIR transposable element annotation, provides evidence for abundant new transposable elements in the maize genome.
Mol Plant, 12(3):447–460.

Wang, X., Yu, B., Ma, A., Chen, C., Liu, B., and Ma, Q. (2018).
Protein–protein interaction sites prediction by ensemble random forests with synthetic minority oversampling technique.
Bioinformatics, 35(14):2395–2402.

References

Yu, J., Shi, S., Zhang, F., Chen, G., and Cao, M. (2019).
PredGly: predicting lysine glycation sites for homo sapiens based on XGboost feature optimization.
Bioinformatics, 35(16):2749–2756.

Zeng, X., Zhong, Y., Lin, W., and Zou, Q. (2019).
Predicting disease-associated circular RNAs using deep forests combined with positive-unlabeled learning methods.
Brief Bioinform.

Zhang, L., Yu, G., Xia, D., and Wang, J. (2019).
Protein-protein interactions prediction based on ensemble deep neural networks.
Neurocomputing, 324:10–19.

Zhang, X., Wang, J., Li, J., Chen, W., and Liu, C. (2018).
Crlncrc: a machine learning-based method for cancer-related long noncoding rna identification using integrated features.
BMC Med Genomics, 11(Suppl 6):120.

References

Zheng, R., Li, M., Chen, X., Wu, F.-X., Pan, Y., and Wang, J. (2019).
BiXGBoost: a scalable, flexible boosting-based method for reconstructing gene regulatory networks.
Bioinformatics, 35(11):1893–1900.

Marcel **Turcotte**
Marcel.Turcotte@uOttawa.ca

School of Electrical Engineering and **Computer Science** (EECS)
University of Ottawa