Ensemble Learning

Introductions

- Methods
  - Bagging
  - Boosting
  - Averaging or Voting
  - Stacking
- Pros & Cons
  - Improve accuracy
  - Reduce interpretability

Basic intuition – major vote

- Suppose we have 3 completely independent classifiers with a 70% accuracy
- For a majority vote with 3 classifiers we can expect 4 outcomes:

All three are correct
  0.7 * 0.7 * 0.7
= 0.3429

Two are correct
  0.7 * 0.7 * 0.3
+ 0.7 * 0.3 * 0.7
+ 0.3 * 0.7 * 0.7
= 0.4409

Two are wrong
  0.3 * 0.3 * 0.7
+ 0.3 * 0.7 * 0.3
+ 0.7 * 0.3 * 0.3
= 0.189

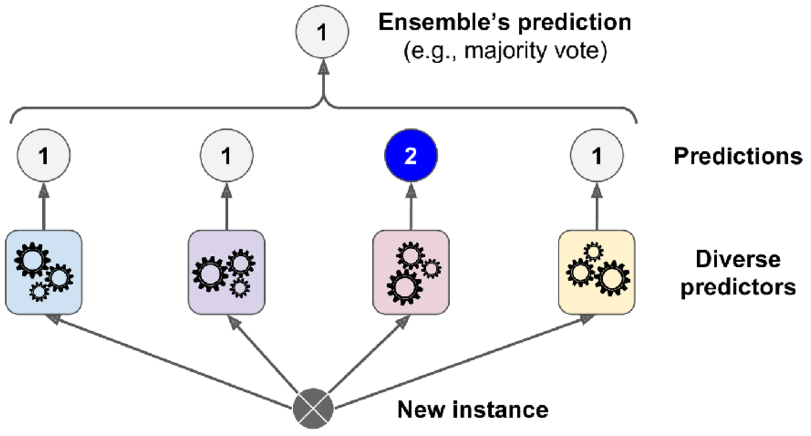
All three are wrong
  0.3 * 0.3 * 0.3
= 0.027

This majority vote ensemble will be correct an average of ~78%
(0.3429 + 0.4409 = 0.7838)

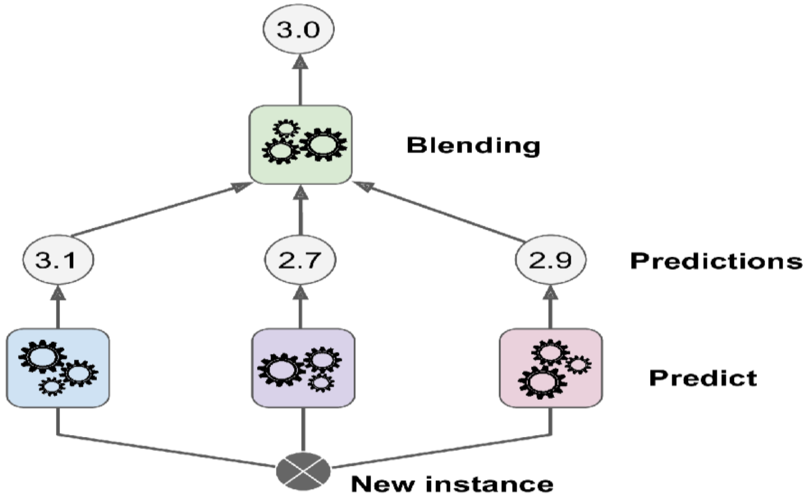
- When we have 5 classifiers (70% accuracy for each)
  - $10x(0.7)^3(0.3)^2 + 5x(0.7)^4(0.3)^1 + (0.7)^5$
  - 83.7% major vote accuracy

Ensemble approaches

- Creating ensembles from predictions or submission files
  - Voting ensemble
  - Averaging



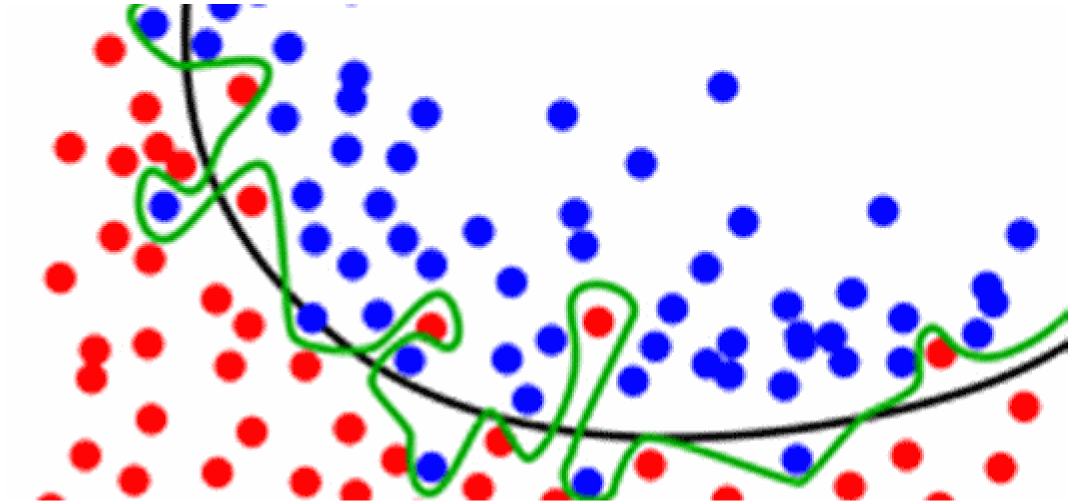
- Stacked Generalization (Stacking)



Source: "Hands-On Machine Learning with Scikit-Learn and TensorFlow" , urélien Géron

The effect of averaging predictions

- You ideally want a smooth separation between classes, and a single model's predictions can be a little rough around the edges.



- The black line shows a better separation than the green line. The green line has learned from noisy data points.
- Averaging multiple different green lines should bring us closer to the black line.

Voting ensemble

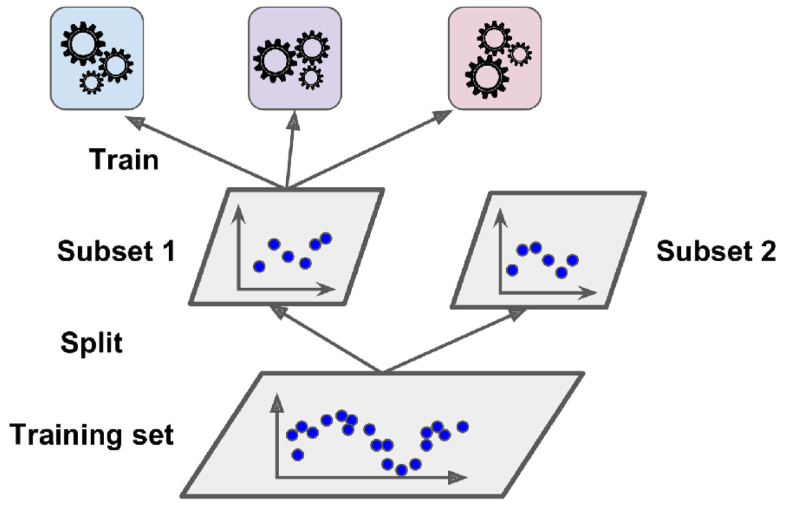
```python
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import VotingClassifier

logreg = LogisticRegression()
tree = DecisionTreeClassifier()
mlp = MLPClassifier()
voting = VotingClassifier(
    estimators = [('logreg', logreg), ('tree', tree), ('mlp', mlp)],
    voting = 'hard')
```

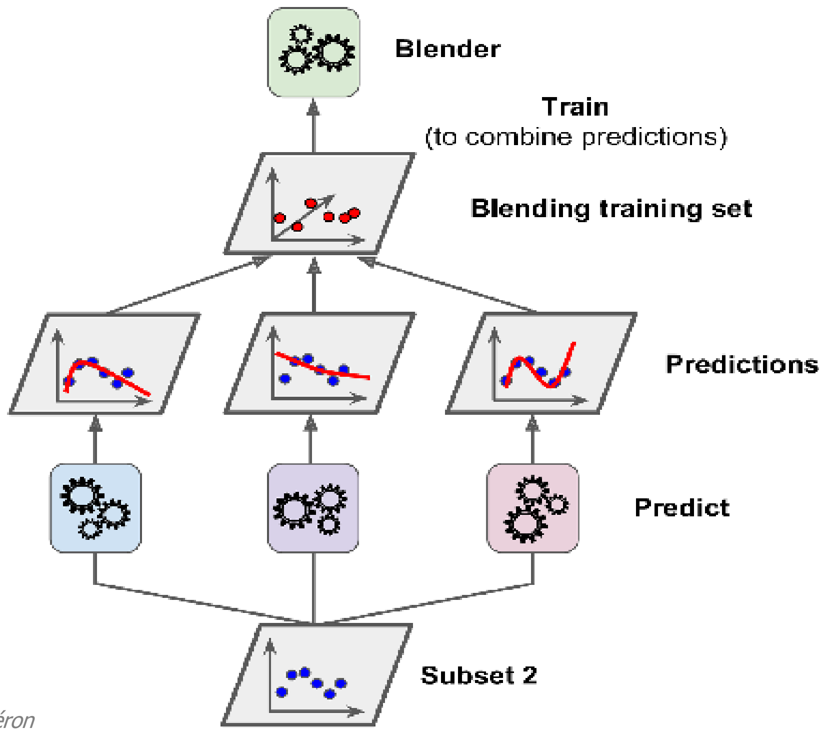
```python
from sklearn.metrics import accuracy_score
for clf in (logreg, tree, mlp, voting) :
    clf.fit(X_train, y_train)
    print(clf.__class__.__name__, accuracy_score(
        y_test, clf.predict(X_test)))
```

```
LogisticRegression 0.9755555555555555
DecisionTreeClassifier 0.9511111111111111
MLPClassifier 0.9888888888888889
VotingClassifier 0.9911111111111112
```

Stacking

- First, the training set is split in two subsets. The first subset is used to train the predictors in the first layer.
- Next, the first layer predictors are used to make predictions on the second (held-out) set.
- A new training set using these predicted values is created as input features. The blender is trained on this new training set.

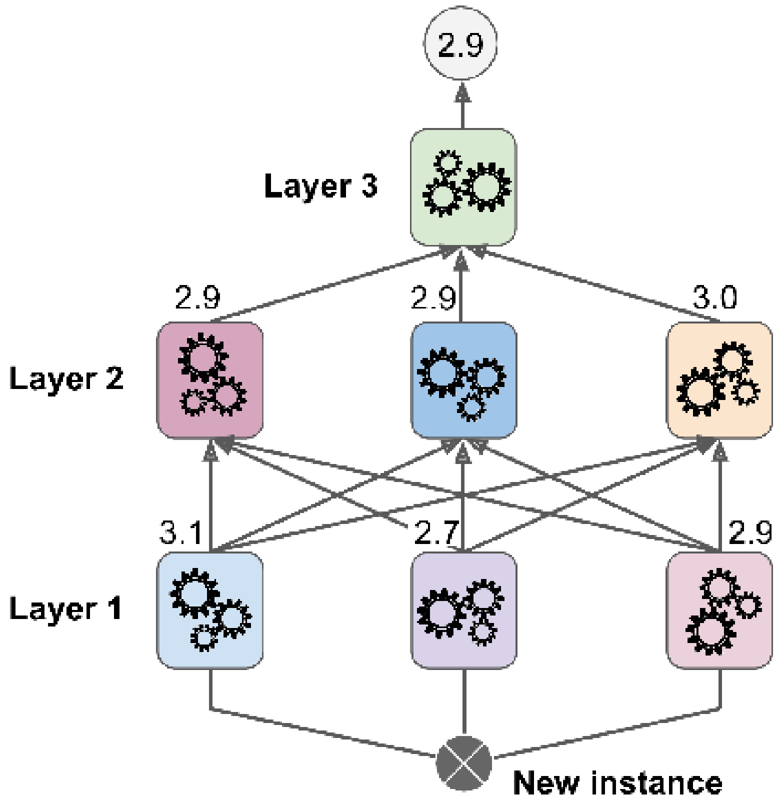




Source: "Hands-On Machine Learning with Scikit-Learn and TensorFlow" , urélien Géron

Stacking

- Multilayer stacking
  - It is actually possible to train several different blenders.
  - The trick is to split the training set into three subsets: the first one is used to train the first layer, the second one is used to create the training set used to train the second layer, and the third one is used to create the training set to train the third layer.
- Related packages
  - scikit-learn does not support stacking.
  - brew, mlxtend, stacking, vecstack



Source: "Hands-On Machine Learning with Scikit-Learn and TensorFlow" , urélien Géron

Stacking

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from mlxtend.classifier import StackingClassifier
```

```
rf = RandomForestClassifier()
tree = DecisionTreeClassifier()
mlp = MLPClassifier()
logreg = LogisticRegression()  # blender or meta-learner
stacking = StackingClassifier(classifiers=[rf, tree, mlp],
                              meta_classifier=logreg,
                              use_probas=False,
                              average_probas=False)
for clf in (rf, tree, mlp, stacking) :
    clf.fit(X_train, y_train)
    print(clf.__class__.__name__, accuracy_score(
        y_test, clf.predict(X_test)))
```

```
RandomForestClassifier 0.9688888888888889
DecisionTreeClassifier 0.9533333333333334
MLPClassifier 0.98
StackingClassifier 0.9822222222222222
```

What is Boosting?

- Ensemble meta-algorithm used to convert many weak learners into a strong learner
- The general idea of most boosting methods is to train predictors sequentially, each trying to correct its predecessor.
- There are many boosting methods available, but by far the most popular is Gradient Boosting
- Gradient Boosting works by sequentially adding predictors to an ensemble, each one correcting its predecessor. This method tries to fit the new predictor to the residual errors made by the previous predictor.

How gradient boosting is accomplished

```
from sklearn.tree import DecisionTreeRegressor

tree_reg1 = DecisionTreeRegressor(max_depth=2)
tree_reg1.fit(X, y)
```

Now train a second `DecisionTreeRegressor` on the residual errors made by the first predictor:

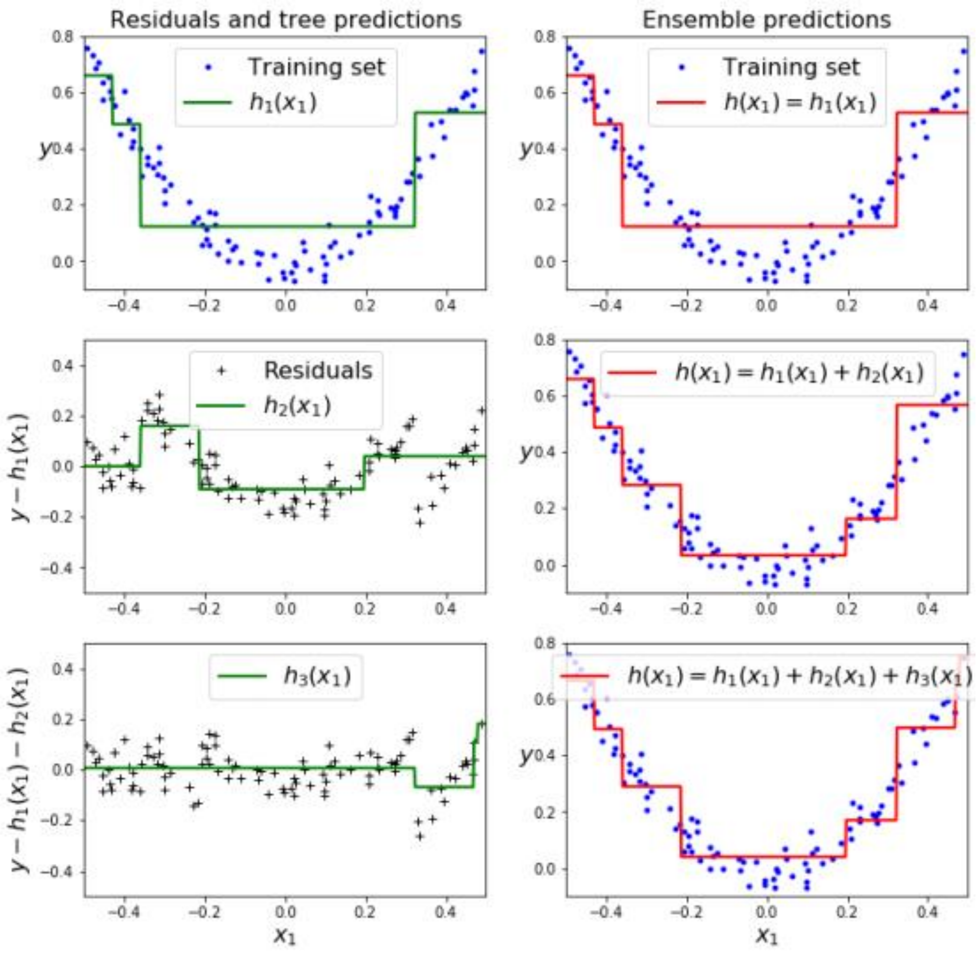
```
y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2)
tree_reg2.fit(X, y2)
```

Then we train a third regressor on the residual errors made by the second predictor:

```
y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2)
tree_reg3.fit(X, y3)
```

Now we have an ensemble containing three trees. It can make predictions on a new instance simply by adding up the predictions of all the trees:

```
y_pred = sum(tree.predict(X_new) for tree in
(tree_reg1, tree_reg2, tree_reg3))
```



Source: "Hands-On Machine Learning with Scikit-Learn and TensorFlow", urélien Géron