Post 1

لو سألنا 1000 شخص عن سؤال ما هنلاقى اجابات مختلفة ولكن تجميع الاجابات ديه هيدينى احسن واقيم حل للسؤال على عكس لو سالت حتى واحد خبير ، ده الى بدء بيه الكاتب فى شابتر 7.
ده يديك intuition من قبل ما تبدء فى الشابتر عن انك كنت بتجرب كذا موديل عشان تشوف الافضل وتستخدمة ، طب ما بدل ما نشوف الافضل هل ممكن نجمع ال predictions ديه كلها وناخد بال Vote !
بمعنى انى لو عندى 5 موديل و 3 منهم عملوا prediction ل class 1 ، والاتنين التانيين عملوا prediction ل class 2 ، ببساطة هاخد class 1 لان ال Vote اعلى بنسبة 3 الى 2 .
الطريقة ديه بتسمى Ensemble وده تقدر تنفذه على مجموعة من ال models المختلفة ، او على نفس ال model تعمله run على subsets مختلفة من الداتا .
والافضل طبعا هيكون مجموعة من ال models المختلفة ، وده لان ال errors هتكون مختلفة من model للتانى على عكس ما استخدم نفس ال model مع قيم مختلفة لل Hyper Parameters.
وهنا بيكون النتيجة افضل من ال best classifier حتى لو كمان كانت ال classifier كلها ضعيفة ده هيدى نتيجة افضل .

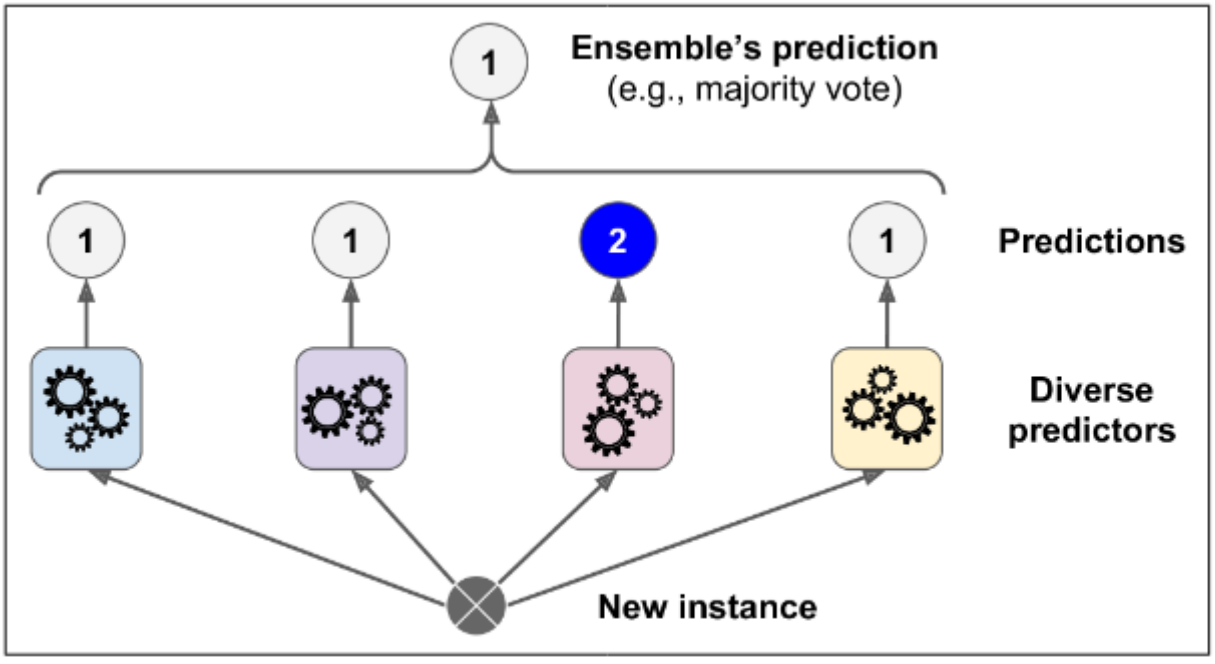

Figure 7-2. Hard voting classifier predictions

Post 2

عملية ال Vote نفسها فيها نوعين ال default هو ال Hard Voting وهو انك زى ما فى الصورة بتشوف كل model عملت prediction لانهى class وتروح تاخد ال Highest Vote.

لكن فى طريقة تانية بتدى نتائج أفضل ، وهو ال Soft Voting ، ده فى حالة انى ال models المستخدمة فيها Probability Prediction ، وهنا بعد ما نجيب ال probability ، نبتدي ناخد ال Average لكل class ، بمعنى انى لو عندى زى ما فى الصورة 4 موديل وكان عدد ال classes هو 2 كده المفروض انى كل class هيكون ليه 4 قيم ، لو جمعناهم وقسمنا على 4 ، كده فى الاخر هيكون عندى قيمتين بيعبروا عن ال overall probability across models ، ومن هنا تقدر تعمل classification to 1 or 0 based on threshold .

فى الصورة ال accuracy فى الطريقتين .

In [14]:
```
print("Hard Vote", accuracy_score(y, y_pred))
print("Soft Vote", accuracy_score(y, overall_prob))
```

```
Hard Vote 0.718
Soft Vote 0.829
```

Post 3

بعد ما اتكلمنا عن الفرق بين ال Soft and Hard vot هنتكلم عن طريقتين اسمهم Bagging and Pasting.

الفكرة هنا من الطرق ديه انك بدل ما كنت بتعمل run لموديلز مختلفة ، هتبتدى تعمل run لنفس الموديل ولكن على subset مختلفة من ال data ، والفرق بين الطريقتين هو انك لما تعمل run لنفس ال model هل بتغير ال subset الى بتعمل عليها run ولا لا ، الاول بيحصل تبديل فى ال samples التانى لا.

ولكن المميز هنا انى ال Baggin Classifier بيعمل soft vot لو كان الموديل المستخدم ليه Predict probability method.

كل ال Voting Classifier, Begging and Pasting نقدر اننا نعملهم run على اكتر من CPUs او server مختلفة لانك بترن اكتر من موديل وكل موديل مستقل عن الموديلز التانية ، وده بيساعد كتير جدا فى ال Training .

فى الصورة مثال انك ترن 500 Decision Tree باستخدام كل ال Cpus المتاحة ، وكل tree هتعمل run على 100 مثال من الداتا ، والفرق بين ال Baggin and Pasting هو تغير متغير ال bootstrap to False .

لو بصينا لموضوع ال Sampels ده هنلاقى مشكلة بتحصل هو انى بعض ال models هتشوف instance اكتر من مرة بينما البعض الاخر مش هيستخدم خالص وده اسمه OOB هنتكلم عليه فى البوست الجاى بإذن الله .

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

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(), n_estimators=500,
    max_samples=100, bootstrap=True, n_jobs=-1)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

Post 4

كنا اكلمنا اخر حاجة عن ال Bagging and Pasting ، وقولنا انى بما انى بعمل run لنفس الموديل على subset مختلفة من الداتا، فهنا فيه بعض ال instance نفس الموديل هيشوفها اكتر من مرة ، بينما البعض الاخر الموديل عمرة ما هيشوفها.

وهنا ده بيسموه Out-Of-Bag Evaluation بمعنى انى ال instance الى هى OOB ديه ممكن استخدمها فى انى اعمل Evaluation عليها بدل ال Validation set ودة ممكن يتحقق ببسياطة عن طريق ال Param بتاع oob_score ب True ، وتقدر بقا تشوف ال score بتاع ال Evaluation ده عن طريق param هو oob_score_ ، كمان لو كان الموديل برضه المستخدم ليه probability method تقدر تستخدم oob_decision_function_ عشان تشوف ال probabilty بتاعت كل instance .

احنا استخدمنا هنا مع الطرق ديه Decision Tree Model وكنا بنقول روح رن 500 مثلا منها ، لكن فيه عندى Random Forest وهى عبارة عن Ensemble method of Decision Tree .

بمعنى انى بدل ما اروح اعمل Ensemble from Decision Tree ممكن اروح علطول استخدم ال Random Forest وده الى هنتكلم عنه البوست الجاى إن شاء الله .

Post 5

قبل ما نبدء كلام فى ال Random Forest Tree ، نرجع لل Bagging and Pasting ، احنا لما كنا بندرب نفس الموديل ولكن على subset مختلفة ، كمان احنا نقدر نعمل ده مع ال features نفسها ، عن طريق max_features and bootstrap_features params ، لو انا عملت sampel للاتنين ال instances and features ده بيسمى Random Patches ، ولو عملت sampel لل Features فقط ده بيسمى Random Subspaces .

نيجى بقا لل Random Forest هو ببساطة انى بدل ما اروح اعمل Bagging classifier واروح اديلة ال Decision Tree ، انا ممكن اروح اعمل Random Forest الى هى عبارة عن Ensemble method من ال Decision Tree ، ال Random Tree ديه فيها معظم ال Hyper params الى موجودة سواء فى ال Decision Tree or Bagging Classifier ، ولكن فى ال Random Forest بتروح تدور على Best feature الى تقدر منه تعمل split فى random subset من ال features ، بدل ما كان بيحصل فى ال Decision Tree انى اروح ادور فى كل ال Features وده طبعا بيوفر وقت كبير ، مش بس كده فى نوع تانى منها اسمه Extra-Randomized-Tree وده بدل ما يروح يشوف Best feature in random subset من ال features لا ده بيختار random threshold for each feature ودة اسرع بكتير من ال Trees الاخرى .

كمان ال RandomForest بتقدر بعد ال Training تعرف انهى Features ليها تاثير اكبر عن طريق متغير feature_importances_ وده يساعدك إنك تعمل features section .

Post 6

الطرق الى فاتت كلها من VotingClassifer, Bagging, Pasting and RandomForest كنت تقدر تشتغل عليها بالتوازى ، لان كل model شغال لوحده وفى الاخر انا بجمع النتائج ديه لما اجى اعمل prediction.

لكن فيه Approach تانى ، هو انك بدل ما تعمل run لل models ديه منفصلة ، نروح نعمل run ل model وبعد كده نعمل prediction على الداتا الى عمل عليها run ديه ونشوف هو وقع في انهي instances يعنى what the model misclassified ، ونروح نعمل updates لل weights المتعلقة بال instances ديه ، وبعدين نروح نعمل train بعد ما عملنا update ، الطريقة ديه من ال Train اسمها Adaptive Boosting ، ولكن للاسف ده مش هينفع انك تعمله run غير Sequential ، لان كل موديل بيروح يعمل correct للموديل الى قبلية وهكذا ، مش بس كده انا هنا بقا عندى weights خاصة بكل instance وبروح اعمل updates لل Weights الخاصة بال misclassified instance ، بدل ما كان عندى shared weights بحاول اعملها optimize بحيث انها تتناسب مع ال data .

بعد كده بيحصل زى ما كان بيحصل وهو فكرة ال Votting ، ولكن هنا كل prediction بيحصل بيكون weights بتاعته مبنية على الموديل الى قبلها بحيث انها تدى نتيجة افضل ، وفى الاخر بشوف ال majority of vote.

الطريقة الاخرى وهى ال Gradient Bossting وهو انى بدل ما اروح اعمل update لل weights الخاصة بال instances الى حصل عليها wrong prediction ، لا انا هروح اعمل update لل weights بس على ال error المتبقى ، بمعنى انى هروح اعمل prediction باول موديل انا عملته ، بعد كده اروح اشوف ال actually y - prediction y_hat وهنا هيجبلى كل ال wrong classified اروح اديها لل model الجديد وهو بيعمل fit زى ما واضح فى الكود .

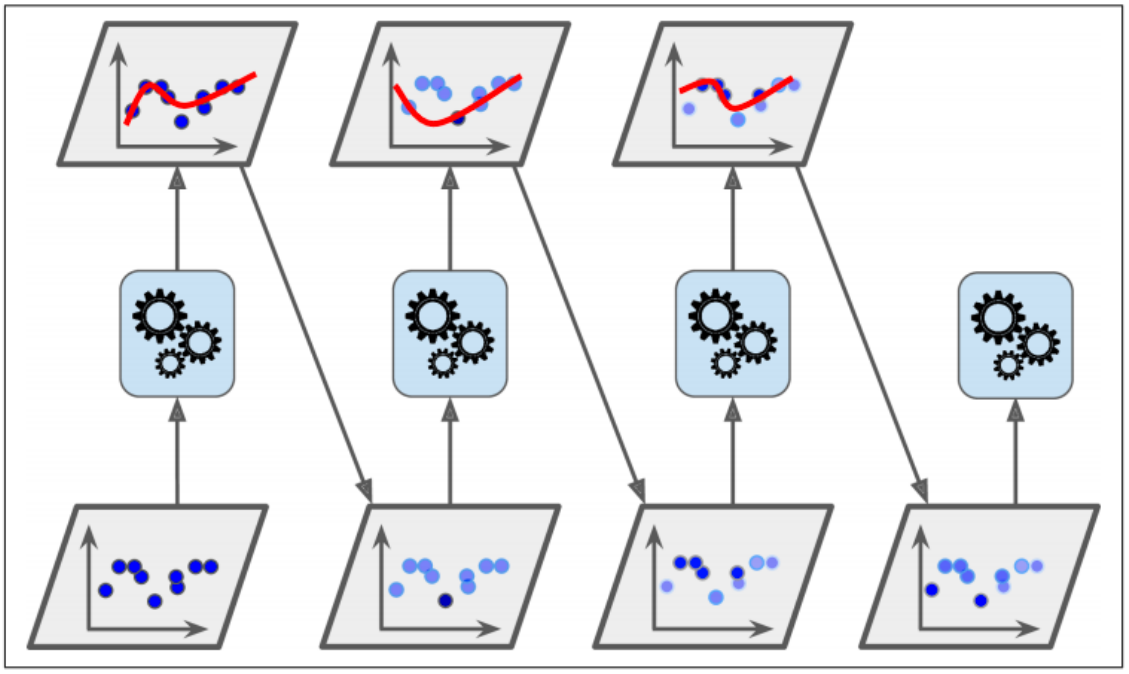

Figure 7-7. AdaBoost sequential training with instance weight updates

```
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))
```