Quantitative Applications for Data Analysis: ensemble methods

Erik Spence

SciNet HPC Consortium

7 April 2022

SciNet compute • calcul CANADA

Today's slides

Today's slides can be found here. Go to the "Quantitative Applications for Data Analysis" page, under Lectures, "Ensemble methods".

https://scinet.courses/1211

SciNet compute • calcul CANADA

Today's class

Today we're going to explore ensemble methods:

- Voting,
- Stacking,
- Bagging,
- Boosting.

These are algorithms that combine models together in the hope of building an even more-awesome model.

Ask questions!

SciNet compute • calcul CANADA

Ensemble methods

It's possible to bundle multiple machine-learning, or regression, models together into a single model. Such techniques are known as "ensemble methods", and the results "ensemble models". Why would we want to do that? What's the point?

- Some methods are stronger on certain data sets than others.
- Different algorithms have different weaknesses or strengths than others.
- By combining the models together we hope to have the different models complement each other, resulting in an aggregate model which is more accurate than the individual models themselves.

We will focus on classification examples today, but many of these models also have regression versions, which could be used instead.

SciNet compute • calcul CANADA

Voting

The simplest of the ensemble methods are

- voting, for classification problems,
- or averaging, for regression problems.

In these methods, multiple classification or regression models are trained. These models can be created by

- using different algorithms (kNN, decision tree, etc),
- using different splits of the training data set,
- using different sets of features from the training data set,
- combinations of the above.

We then generate the predictions for the test data for all the models.

SciNet compute • calcul CANADA

Majority voting

The simplest voting method is majority voting.

- For each test data point the final result is the result which gets more than half of the votes from the various models.
- If no prediction gets more than half of the votes, we say that the ensemble method could not make a stable prediction.
- In this case we would usually pick the prediction that gets the most votes. This is sometimes called "plurality voting".
- Another example is weighted voting: the votes from the different models are weighed unequally. The prediction with the highest weighted value of votes is the winner.

For continuous models the predicted values are either simply averaged, or a weighted average is used.

SciNet compute • calcul CANADA

Voting, example

Let's use something new, the forest-cover-type data set.

- The goal of the data set is to predict the type of forest cover, based on 54 different features.
- There are 7 different forest-cover types.
- The data set is large, it may take a minute to download the first time.
- Because the data set is so large, we'll only select a small subset of the whole.

```
In [1]:
In [1]: import skearn.model_selection as skms
In [2]: import sklearn.preprocessing as skpp
In [3]: import skearn.datasets as skd
In [4]:
In [4]: data = skd.fetch_covtype()
In [5]:
In [5]: train_x, test_x, train_y, test_y = \
...:        skms.train_test_split(data.data,
...:                              data.target,
...:                              test_size = 1000,
...:                              train_size = 5000)
In [6]:
In [6]: train_x.shape
Out[6]: (5000, 54)
In [7]:
```

Voting, example, continued

The voting model is built into sklearn.

- The model is assembled as a list of tuples. Each tuple consists of a label and model.
- You can put in as many models as you like.

Support Vector Machines are a geometric algorithm so we need to scale and centre the data. For this we use a pipeline. Logistic regression seems to benefit for this data too.

```
In [7]: import sklearn.tree as skt, sklearn.svm as svm
In [8]: import sklearn.linear_model as sklm
In [9]: import sklearn.pipeline as skp
In [10]: import sklearn.ensemble as ske
In [11]:
In [11]: tree_model = skt.DecisionTreeClassifier()
In [12]: lr_model = skp.make_pipeline(
...:          skpp.StandardScaler(),
...:          sklm.LogisticRegression(max_iter = 10000))
In [13]: svm_model = skp.make_pipeline(
...:                  skpp.StandardScaler(), svm.SVC())
In [14]:
In [14]: voter = [("DT", tree_model)]
In [15]: voter.append(("LR", lr_model))
In [16]: voter.append(("SVM", svm_model))
In [17]: voting_model = ske.VotingClassifier(voter)
```

Voting, example, continued more

Note that ensemble models will often get better answers than individual models, but not always.

As we've seen before, the results will depend upon the train/test split. And of course, we are looking at the training data here, not the test data.

```
In [18]:
In [18]: def test_models(models, labels, x, y):
...:         for model, label in zip(models, labels):
...:             scores = skms.cross_val_score(model,
...:                         x, y, cv = 10)
...:             print(label, scores.mean())
In [19]:
In [19]: labels = ['DT', 'SVM', 'LR', 'Voting']
In [20]: models = [tree_model, svm_model, lr_model,
...:               voting_model]
In [21]:
In [21]: test_models(models, labels, train_x, train_y)
DT 0.689
SVM 0.729
LR 0.7152
Voting 0.7384
In [22]:
```

Voting, example, continued even more

In this case the voting model does not do better than the straight-up SVM model, on the test data (and the training data, as it happens).

But of course, we can't choose our optimal model based on the test data, but rather we must use the training data (hence the need for cross-validation).

```
In [22]: import sklearn.metrics as skm
In [23]:
In [23]: voting_model = voting_model.fit(train_x,
...:                                     train_y)
In [24]: voting_pred = voting_model.predict(test_x)
In [25]: skm.accuracy_score(test_y, voting_pred)
Out[25]: 0.757
In [26]:
In [26]: svm_model = svm_model.fit(train_x, train_y)
In [27]: svm_pred = svm_model.predict(test_x)
In [28]: skm.accuracy_score(test_y, svm_pred)
Out[28]: 0.747
In [29]:
```

Stacking

Stacking involves the building of a meta-model from some base models.

- We first train a bunch of models on the training data set.
- These models are then used to predict the targets of the training data.
- We then create a new data set, consisting of these target *predictions* from the trained models, and the correct target from the original training data set.
- We then train a meta-model on this new data set.
- By using the predictions of the different models as features, we can let the meta-model determine when certain models perform well, and when certain models perform poorly.

This technique is particularly useful for combining models of different types.

SciNet compute • calcul CANADA

Stacking, example

Let's continue our previous forest-cover example using stacking.

Note that we need to specify the base models and the meta-model which is fit to the created data set.

```
In [29]:
In [29]: import sklearn.neighbors as skn
In [30]:
In [30]: knn1_model = skp.make_pipeline(
...:             skpp.StandardScaler(),
...:             skn.KNeighborsClassifier(1))
In [31]: stack = [("kNN1", knn1_model)]
In [32]: stack.append(("DT", tree_model))
In [33]: stack.append(("SVM", svm_model))
In [34]: stack.append(("LR", lr_model))
In [35]:
In [35]: stacked = ske.StackingClassifier(
...:       estimators = stack,
...:       final_estimator =
...:         sklm.LogisticRegression(max_iter = 10000))
In [36]:
```

Stacking, example, continued

The stacking model is trained the same way you train a regular sklearn model. You can even run cross-validation on it.

Note that stacking models can be slow to train. There are many models to train.

Note again that the ensemble method in this case outperforms the base models.

```
In [36]:
In [36]: labels = ['kNN1', 'DT', 'SVM', 'LR', 'Stacking']
In [37]: models = [knn1_model, tree_model, svm_model,
...:                       lr_model, stacked]
In [38]:
In [38]: test_models(models, labels, train_x, train_y)
kNN1 0.7258
DT 0.689
SVM 0.729
LR 0.7152
Stacking 0.755
In [39]:
```

A note on performance

As has been mentioned already, not all ensemble models will outperform the models from which they are built.

- Sometimes the underperforming base models drag the ensemble model down with them.
- Sometimes the choice of base models or meta-models is not appropriate for the data set in question.
- Certain data sets lend themselves better to different types of base models; the same applies to ensemble methods.
- That being said, many of the latest machine-learning-competition-winning models have been ensemble methods.
- If you go down this road you will need to experiment with many combinations of base models and parameters to find the best-performing combination.

We will stick with the forest cover data set for the rest of this class. Some models will do better on this data set than others.

Bootstrap aggregating (bagging)

Another ensemble technique is Bootstrap Aggregating (commonly known as "Bagging").

- This is useful for algorithms that have a high variance (decision trees).
- We first train a bunch of models of the same type on different versions of the training data set.
- These data set versions are generated using bootstrapping (sampling from the training data with replacement).
- These models are then aggregated using voting (classification) or averaging (regression).
- The many models can be generated in parallel.

This is a very commonly-used technique if you have a base model that you're confident in.

SciNet compute • calcul CANADA

Bagging, variations

You may run into a variety of different variations on Bagging:

- "Pasting": samples are drawn from the data set without replacement. This was originally designed for large data sets.
- "Bagging": samples are drawn from the data set with replacement (the data is bootstrapped). This is similar to non-parametric bootstrapping.
- "Random Subspaces": Each model is trained on a subset of the features. Also known as "feature bagging".
- "Random Patches": the combination of Bagging and Random Subspaces, the models are trained on subsets of both the samples and the features.

Both Bagging and Random Patches are worth exploring if you end up going down this road.

SciNet compute • calcul CANADA

Bagging, example

Since kNN is the best individual model so far, let's see if Bagging can improve upon it.

Bagging is built into sklearn. We can use optional flags to convert from Bagging to Random Patches:

- n_estimators: the number of instances of the model to train.
- max_samples: the number, or fraction, of data to use.
- max_features: the number, or fraction, of features to use.

```
In [39]:
In [39]: bag = ske.BaggingClassifier(
...:         base_estimator = knn1_model,
...:         n_estimators = 10,
...:         max_samples = 0.8, max_features = 0.8)
In [40]:
In [40]: models = [knn1_model, stacked, bag]
In [41]: labels = ['kNN1', 'Stacking', 'bag']
In [42]:
In [42]: test_models(model, labels, train_x, train_y)
kNN1 0.7258
Stacking 0.755
bag 0.72539999999999
In [43]:
```

Random Forests

Random Forests are a special variation of bagging, based entirely on decision trees.

- As with regular bagging, the data used in training are sampled from the training data, with replacement.
- But rather than allow the tree to split on all available features, only a randomly-chosen subset of the full set of features is available at each split.
- This is actually closer to Random Patches:
  - Random Patches use a subset of features for each model.
  - Random Forests use a subset of features for each split, but the model itself has access to all features.
- Each tree is grown as far as possible, or close to it, without pruning.
- The results of all the trees are then aggregated.

This reduces the correlation between individual models and the high variance which is inherent to decision trees.

SciNet compute • calcul CANADA

Random Forests, example

Random Forests are in the sklearn ensembles subpackage.

- n_estimators: the number of instances of the model to train.
- max_features: the number, or fraction, of features to consider at each split. By default this is $\sqrt{n_{\text{features}}}$.

```
In [43]:
In [43]: tree_model = skt.DecisionTreeClassifier()
In [44]:
In [44]: forest_model = ske.RandomForestClassifier(
...:            n_estimators = 50)
In [45]:
In [45]: models = [tree_model, stacked, forest_model]
In [46]: labels = ['DT', 'Stacking', 'RF']
In [47]:
In [47]: test_models(models, labels, train_x, train_y)
DT 0.6902
Stacking 0.7556
RF 0.7792
In [48]:
```

Extra Trees

A variation on Random Forests is Extremely Randomized Trees (Extra Trees). These are like Random Forests, except

- don't sample the data with replacement, use the whole data set.
- when splitting on continuous features, the split location is chosen randomly.

Sometimes (not always) this will lead to an improvement over Random Forests. This is especially true with noisy, high-dimensional data.

```
In [48]:
In [48]: extra_model = ske.ExtraTreesClassifier(
...:             n_estimators = 50)
In [49]:
In [49]: models = [tree_model, stacked, forest_model,
...:                   extra_model]
In [50]: labels = ['DT', 'Stacking', 'RF', 'ET']
In [51]:
In [51]: test_models(models, labels, train_x, train_y)
DT 0.6838
Stacking 0.7552
RF 0.7714
ET 0.7778
In [52]:
```

Boosting

Boosting is used to convert weak models into strong models.

- In this case, "weak" means a poor classification rate.
- The skeleton of the algorithm is as follows:
  1. Start with a starting model, and the whole data set.
  2. Train the existing model on the remaining data.
  3. Remove the data which the model gets correct.
  4. Create a new model; train it on the remaining data (the data the model gets wrong).
  5. Aggregate the new model with the existing models, using weighted majority vote (classification) or weighted sum (regression).
  6. Repeat, starting at step 3.
- The algorithm actively attempts to correct for mistakes in the existing model.

Adaboost (adaptive boosting) was an early example of this type of boosting algorithm.

SciNet compute • calcul CANADA

Boosting, example

The Adaboost model is built into sklearn.

You can specify the type of base model to use to build the full model. The default is a decision tree.

Though Adaboost is a useful algorithm, it's been succeeded by Gradient Boosting, and so is not as widely used as it once was.

```
In [52]:
In [52]: tree_model = skt.DecisionTreeClassifier()
In [53]:
In [53]: ada_model = ske.AdaBoostClassifier(
...:             base_estimator = tree_model,
...:             n_estimators = 30)
In [54]:
In [54]: models = [tree_model, ada_model]
In [55]: labels = ['DT', 'Ada']
In [56]:
In [56]: test_models(models, labels, train_x, train_y)
DT 0.691
Ada 0.6908
In [57]:
```

SciNet compute • calcul CANADA

Gradient Boosting

Gradient Boosting (also called Accelerated Gradient Boosting, or Gradient Tree Boosting) is a generalization of boosting. It's based on three parts:

- A loss function. This depends on the problem type, but must be differentiable.
- A (weak) model, usually a decision tree.
- A meta-model, which combines the weak models to minimize the loss function. The loss function is minimized using gradient descent.

$$\boldsymbol{F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)}$$

where $\boldsymbol{m}$ is the iteration step. The model is built sequentially. At each step the decision tree $\boldsymbol{h_m(x)}$ is chosen to minimize the loss function $\boldsymbol{L}$, given the current model $\boldsymbol{F_{m-1}(x)}$.

$$\boldsymbol{F_m(x) = F_{m-1}(x) + \mathbf{argmin}_h \sum_i^n L(y_i, F_{m-1}(x_i) + h(x_i))}$$

XGBoost

If you start using Gradient Boosting, you'll quickly run into XGBoost ("eXtreme Gradient Boosting"). This is a very popular modelling algorithm.

- XGBoost is basically the same as the GradientBoosting model built into sklearn.
- However, there are important differences, under the hood, between XGBoost and sklearn's GradientBoosting:
  - XGBoost is fast; it was designed for speed.
  - XGBoost can be run in parallel, either single- or multi-node.
  - XGBoost uses less memory.
- However, XGBoost does not come with sklearn. You need to install the "xgboost" package.

SciNet compute • calcul CANADA

XGBoost, example

The XGBoost package must be downloaded and installed separately.

Since the Extra Trees model is the current best we've seen, we'll compare against that.

Because the forest-cover-type data set is so large this example takes quite a bit of time to run.

```
In [57]:
In [57]: import xgboost as xgb
In [58]:
In [58]: xgb_model = xgb.XGBClassifier()
In [59]:
In [59]: models = [stacked, forest_model,
...:               extra_model, xgb_model]
In [60]: labels = ['Stacking', 'RF', 'ET', 'XGB']
In [61]:
In [61]: test_models(models, labels, train_x, train_y)
Stacking 0.7562
RF 0.7712
ET 0.7758
XGB 0.7638
In [62]:
```

XGBoost, example, continued

Out of curiosity, we might look at the results of the test data, just to see.

Note, of course, that we can't pick our model based on the results of the test data. We should use the results of the cross-validation to pick the final model.

```
In [62]:
In [62]: extra_model = extra_model.fit(train_x, train_y)
In [63]:
In [63]: xgb_model = xgb_model.fit(train_x, train_y)
In [64]:
In [64]: skm.accuracy_score(test_y,
...:                         extra_model.predict(test_x))
Out[64]: 0.8
In [65]: skm.accuracy_score(test_y,
...:                         xgb_model.predict(test_x))
Out[65]: 0.779
In [66]:
```

Summary

Some things to remember:

- Ensemble models combine a bunch of other base models to (hopefully) create a more-accurate model than the base models themselves.
- Voting is the simplest approach, either doing majority voting or averaging of results.
- Stacking involves creating a meta-model which models the results of the base models, hopefully learning where certain base models are weak.
- Bagging creates multiple versions of the same base model, but each trained on different boostrapped versions of the original data set.
- Random Forests is a tree-based versions of bagging, where only subsets of features are available at each split. Extra Trees is similar, but when splitting on continuous features the splits are chosen randomly.
- Gradient Boosting uses Gradient Descent to iteratively build a better model, by focussing on what the model gets wrong. XGBoost is a super version.

SciNet compute • calcul CANADA