ENSEMBLE LEARNING

End to End Understanding

Anish Mahapatra

Contents

Introduction

Data Science replicates human behaviour. We have designed machine learning to imitate how we behave as humans. Think of a model in Data Science as one way to learn. Human beings have a bias when we make a choice. The way one person lives their life cannot be scaled across the human race. Instead, when multiple people share their experiences and learnings, it is possible to develop a generalised approach. It is the same with models in Data Science. One model can do a good job with a machine learning problem, but a set of models will do a better job in most cases. This enhanced performance is because the combined model is more generalised with less bias.

Let's say we are trying to get fit by controlling our diet. Your mother might suggest *reducing the quantity* of food you eat, but make sure that you eat breakfast, lunch and dinner. You might have friends that say that *intermittent fasting* is the best way to go about it. Another friend might suggest a *keto diet* to optimise the number of macronutrients your body is getting. So, in theory, all of these are the possible models that might work for your use case. The question is, are you going to trust one model? Perhaps a better approach might be some combination of these suggestions that work best for you.

That's it. That is the concept of **ensemble models**.

It is important to note that only the best models don't need to be used in an ensemble model. Instead, it should work for the problem at hand that we are trying to solve.

Some of the best models that are implemented on Kaggle are ensemble models. It makes people uneasy when they do not know much about this topic. Read this blog. Read it all the way through, and whether you are a business user or a tech person or even just curious, you will understand models end to end. Do not hesitate to reach out to us if you want to learn how to become a Data Scientist or perfect the tools that a Data Scientist uses **in the real world**.

1 What is Ensemble Learning?

Ensemble Learning is the process where various models are combined to get better results.

The concepts that we will discuss are easy to grasp. From the introduction, we have an intuition about Ensemble models. The core idea is that the result obtained from a combination of models can be more accurate than any single member in the group.

Let's go over the previous example. We were discussing what the best method to get fit by controlling our diet is. Essentially, all of the suggested methods are ways to optimise calorie intake. Each method has different yet effective ways of getting good results. Combining the processes in the right way will help get the most out of the entire exercise.

Imagine a model to be a person who analyses patterns and suggests what you can do next. In the example we have taken, the advice from each person can be considered a model, based on their previous experience, indicates the best way to get fit. Models can also be called learners. Each

learner is considered "weak" on their own. However, when multiple weak learners are combined in some form or the other, it forms a robust model. This strong model is what we will be calling an ensemble model. The models should be as diverse as possible. Each model might perform well on different portions of the data. When the models are combined, they cancel out each other's weaknesses. The idea of strength in multiple models over a single one is the premise of ensemble models.

2 Why use Ensemble Learning?

In short, we use ensemble models because they tend to give better results. It's unfair to put the pressure of getting the best possible result on a single model. Even if the model provides good results on a particular dataset, we might see varied performance on other datasets.

Go back to college. You have some great teachers and some who are not. Now, let's say that you are trying to maximise your marks in discrete mathematics. There are three teachers – one who teaches the best and has the best *video lectures*, one who hints at the *questions that will come in exams* and one who gives the *best notes*. Choosing any one of these teachers has advantages and disadvantages. The most optimal approach is to apply an ensemble method to this scenario and mix the offerings by all three teachers to optimise your marks in discrete mathematics. It works the same way for ensemble models. You have now understood why we use ensemble models and why they are preferred over individual models in many scenarios.

All right, it's been fun understanding the ins and outs of ensemble modelling. Now, let's get to business. Even though I am not a fan of learning Data Science using jargons, it is the language the world prefers and understands.

I want to reiterate that the flow I prefer in the below topics provides an **intuition**, followed by **business understanding** and then ultimately a **coding** example. Remember, if you would like to work on *solved* end to end Data Science Projects, you can reach out to us for this or anything else here. We are here to help, and you can ask us any questions that are on your mind.

3 Types of Ensemble Methods

In the below sections, we will go over the different types of ensemble models. First, we will go over the fundamental models followed by the more intricate ensemble learning techniques used across Machine Learning. Let's Begin!

3.1 Fundamental Ensemble Models

The below sections will cover max voting, averaging and weighted average methods. These are fundamental yet extremely powerful techniques. It's fun to read and understand Data Science in theory. If you want to learn Data Science, do it. To aid you with this, we will be sharing the Google Colab Notebook link so that you can work alongside us and understand what happens.

3.1.1 Max Voting

Intuition

How does voting work? Voting works because the opinion of the majority holds more weight than the vote on an individual. Max Voting is used when we have discrete options on which the models can take a vote. The option that has the most number of votes is considered the chosen one. It is used for classification problems. Each model makes a vote, and the option with the maximum vote is the selected option.

Business Understanding

You decide to go out with your friends. Any cuisine and restaurant can be chosen. All of your vote and the options with the maximum votes are where you guys go in the end. The idea of a democratic system is the premise of Max Voting. Simple, yet effective.

Logic/ Code:

There are two types of max voting – hard and soft. The code can be further understood from the `VotingClassifier()` documentation here. Let's go through examples of both hard and soft.

Imagine we have three classifiers, 1, 2 and 3 and two classes A and B that we are trying to predict. After training the classifier models, we are trying to predict the class of a single point.

- **Hard Voting**
  Predictions:
  Classifier 1 predicts *Class* ***A***
  Classifier 2 predicts *Class* ***B***
  Classifier 3 predicts *Class* ***B***

  2/3 classifier models predict class B.
  Thus, ***Class B*** is the **ensemble decision**.

- **Soft Voting**
  The use case is identical to the example above. The choices for the classes is now expressed in terms of probabilities. The values shown are only for Class A as the problem is binary.
  Predictions:
  Classifier 1 predicts *Class* ***A*** with the probability of 99%
  Classifier 2 predicts *Class* ***A*** with the probability of 49%
  Classifier 3 predicts *Class* ***A*** with the probability of 49%

  The average probability of belonging to Class A is (99 + 49 + 49) / 3 = 65.67%.
  Thus, ***Class A*** is the **ensemble decision**.

In the above scenario, hard voting and soft voting would give different decisions. Soft voting considers how confident each voter is, rather than just a binary input from the voter.

```
# ProjectPro - https://www.dezyre.com/
# 3.1.1 - Max Voting
# Using sklearn.ensemble.VotingClassifier to perform Max Voting
vc = VotingClassifier(

    # Support Vector Machine, XGBoost and Random Forest Classifier
    estimators=[
        ('svm', best_svm),
        ('xgb', best_xgb),
        ('rfc', best_rfc)
    ],

    # soft = Probabilities are taken into account
    # hard = Only the outcome counts
    voting='soft',

    # You can add weights, if required
    weights=[1, 1, 1]

)

# Fitting
vc = vc.fit(X_train, y_train)
```

Code for Max Voting Ensemble Model (Link)

The `sklearn` voting classifier considers max voting from three models – Support Vector Machine, XGBoost and Random Forest Classifier. We used soft voting with equal weights for all the models. You can go and try it out on the Google Colab Notebook.

3.1.2 Averaging

Intuition

For a data point that we are trying to predict, multiple predictions are made by various models. The average of the model predictions is the final prediction that we consider.

Buiness Understanding

Let's say you are watching a game of cricket on television with your friends, and after considering multiple factors, the score that India will hit is to be predicted. Of course, everyone will have a different guess. A good guess would be an average of everyone's predictions. The stronger the individual guesses, the better the overall ensemble model is. The averaging ensemble model works by taking a simple average.

Logic/ Code

The logic is simple once we have the models that we are going to use. Then, we will be leveraging the model that we have and performing an average of the results.

```
# ProjectPro - https://www.dezyre.com/
# 3.1.2 - Averaging

# Results of XGBoost
resultXGB = best_xgb.best_score_*100
print(f'XGBoost: {resultXGB:,.2f}%')

# Results of SVM
resultSVM = best_svm.best_score_*100
print(f'Support Vector Machine: {resultSVM:,.2f}%')

# Results of Random Forest
resultRFC = best_rfc.best_score_*100
print(f'Random Forest Classifier: {resultRFC:,.2f}%')
```

```
XGBoost: 82.30%
Support Vector Machine: 82.58%
Random Forest Classifier: 82.44%
```

```
# Averaging the results of the three models
resultAverage = (resultXGB + resultSVM + resultRFC) / 3
print(f'Ensemble Model - Averaging: {resultXGB:,.2f}%')
```

```
Ensemble Model - Averaging: 82.30%
```

Code for Averaging Ensemble Model

More information on this can be found in the Google Colab Notebook you can refer to for the code. The code takes the model results from three models and provides the average of the outputs.

3.1.3 Weighted Average

Intuition

A weighted average ensemble model allows multiple models to contribute to the prediction based on how good the model is. If a model does better on the dataset in general, we will give it a higher weight. This generalisation will help reduce bias and improve overall performance.

Business Understanding

Let's take the same example of watching a cricket match and predicting the score as effectively as possible. Let's say you have a friend who has been watching cricket religiously for 20 years. We can take a weighted average, where the friend is assigned a higher weightage than somebody new to cricket. In other words, we will give a higher weight to the friend who is more experienced.

Logic/ Code

```
# ProjectPro - https://www.dezyre.com/
# 3.1.3 - Weighted Average Ensemble Model
# Weight assigned and Result for XGBoost
weightXGB = 0.3
resultXGB = best_xgb.best_score_*100
print(f'Weight for XGBoost: {weightXGB:,.1f}')
print(f'Result for XGBoost: {resultXGB:,.2f}%')
```

```
Weight for XGBoost: 0.3
Result for XGBoost: 82.30%
```

```
# Weight assigned and Result for Support Vector Machine
weightSVM = 0.3
resultSVM = best_svm.best_score_*100
print(f'Weight for Support Vector Machine: {weightSVM:,.1f}')
print(f'Result for Support Vector Machine: {resultSVM:,.2f}%')
```

```
Weight for Support Vector Machine: 0.3
Result for Support Vector Machine: 82.58%
```

```
# Weight assigned and Result for Random Forest Classifier
weightRFC = 0.4
resultRFC = best_rfc.best_score_*100
print(f'Weight for Random Forest Classifier: {weightRFC:,.1f}')
print(f'Result for Random Forest Classifier: {resultRFC:,.2f}%')
```

```
Weight for Random Forest Classifier: 0.4
Result for Random Forest Classifier: 82.44%
```

```
# Calculating Weighted Average
resultWeightedAverage = ( weightXGB*resultXGB + weightSVM*resultSVM + weightRFC*resultRFC)
print(f'Result for Weighted Average: {resultWeightedAverage:,.2f}%')
```

```
Result for : 82.44%
```

Code for Weighted Average Ensemble Model

Weights are assigned to each model based on certain factors. Here, we have assigned weights to the models and have gotten a weighted average as the output.

3.2 Bagging

Intuition

Bagging stands for **B**ootstrapped **Ag**gregat**ing**. Bootstrapping is to make the best of a situation using existing resources. Aggregating is grouping what we have into a class or cluster. Bagging has two main steps:

- Use our existing train data and make multiple data instances (**bootstrapping**) – here, you can use the same training samples multiple times. We can sample the data with replacement
- Make multiple models from this bootstrapped data and multiple model outputs. **Aggregate** the results of the model and get the final result

Business Understanding

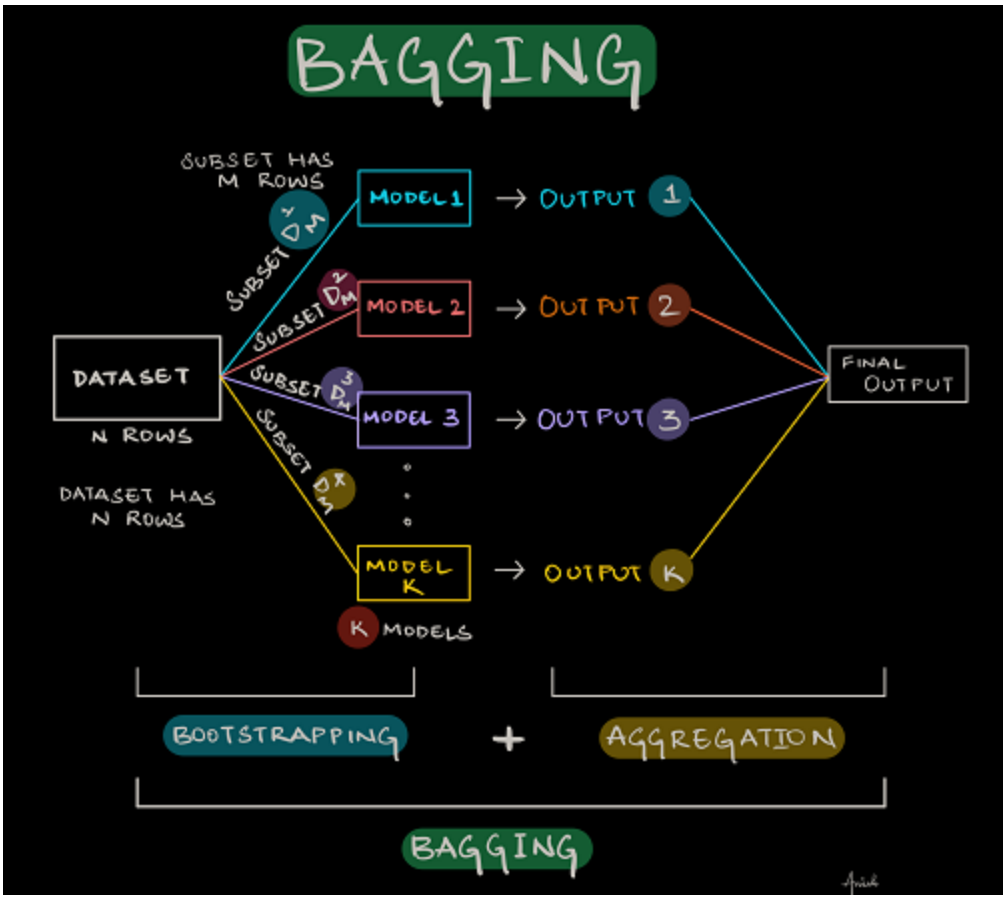
Let's say you are trying to teach kids in fifth grade. Now say you have 50 True or False questions that you would like for them to answer. From this, you decide to teach/ train the kids on 40 questions and test them on ten questions. In this scenario, the 50 questions are the entire dataset, the 40 questions are the train data, and ultimately, the ten questions is the test data.

Let's say you have 25 children. One possible way could be you teach them all together and then test them all together. Here, we will be able to maximise using the 40 train questions that we have. But, each kid has learnt it in the same way.

The new innovative approach gives each of the 25 children a random set of questions (Always less than the 40 train questions). It is okay for questions to repeat. In this case, we are making the best use of the resources we have at hand, called **bootstrapping**.

Next, the kids are going to go ahead and try to answer the 10 test questions. Here, we will let the kids answer true or false and for each of the ten questions. Then, finally, we will take a max vote on the final result. This is called **aggregating**.

In conclusion, we bootstrap the train data we have, make multiple models and aggregate the results. The combination of bootstrapping and aggregating is called bagging.



Visual Representation of Bagging

Logic / Code

Let's get on with a bit of jargon. First, remember what you have learnt, and then try to read the official definition of Bagging.

A Bagging classifier is an ensemble meta-estimator that fits base classifiers on random subsets of the original dataset and then aggregates their predictions (either by voting or by averaging) to form a final prediction.

```
# 3.2 - Bagging Ensemble Model
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import GridSearchCV, StratifiedShuffleSplit

n_estimators = [10,30,50,70,80,150,160, 170,175,180,185];

cv = StratifiedShuffleSplit(n_splits=10, test_size=.30, random_state=15)
parameters = {'n_estimators':n_estimators,}
grid = GridSearchCV(
                    BaggingClassifier
                    (
                    # Default for base estimator is a Decision Tree
                      base_estimator= None,
                    # Indicative of whether a feature is drawn with replacement
                      bootstrap_features=False),
                      param_grid=parameters,
                      cv=cv,
                      n_jobs = -1
                    )
grid.fit(X,y)
```

Code for Bagging Ensemble Model

The sklearn library has the bagging classifier, and more can be found in the official documentation here. This code performs a grid search on the bagging classifier to get the best possible model.

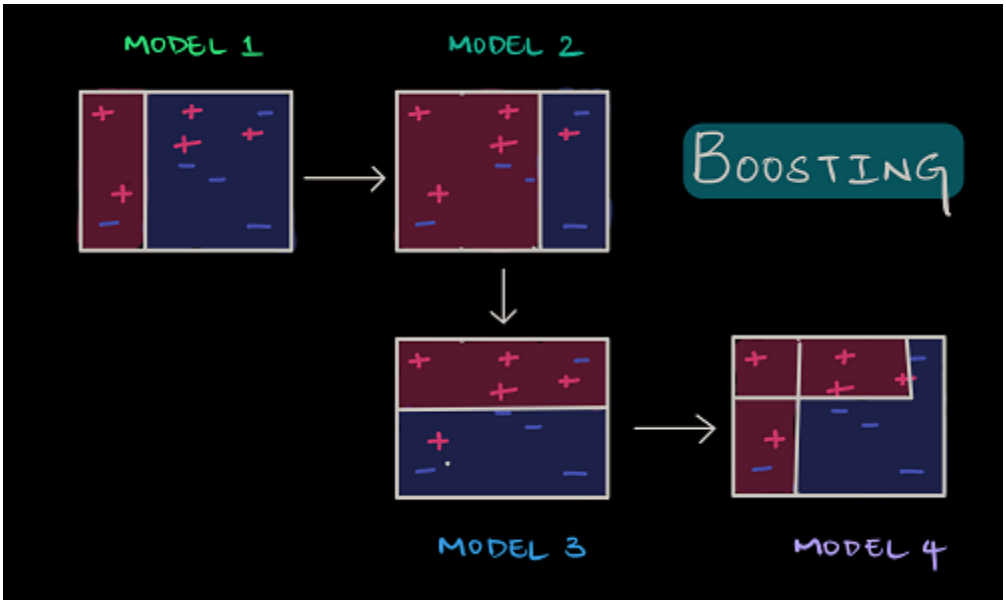
3.3 Boosting

Intuition

This one is fun. The premise of boosting is what it says. First, a model is run, and the results are obtained. Then, specific points are classified correctly, and the rest incorrectly. Now, we will give more weight to the points that are incorrectly classified and rerun the model. What happens? Since there are specific points with a higher weight, the model is more likely to include them. We keep repeating this process, and multiple models are created, where each one corrects the errors of the previous one.

Business Understanding

Say you are teaching a kid a topic in mathematics. They make mistakes as they start, and with each session, you highlight their error so that they do not repeat it. After many sessions, it is clear to the kid how best to differentiate between right and wrong. This gradual improvement was made by highlighting (boosting) their mistakes (like misclassifications, as stated in our previous example).



Visual Representation of Boosting

Logic / Code

Each of the models we make initially has a unique set of learnings. As it is learning, it is called a weak learner in this scenario. Thus, the boosting algorithm combines several weak learners to form a strong learner. There are different kinds of boosting algorithms, and the more popular ones are XGBoost and AdaBoost.

```
# ProjectPro - https://www.dezyre.com/
# 3.3 - Boosting: XGBoost
import xgboost as xgb

# parameter grid for our search
parameters = {
    'min_child_weight': [5, 6, 7],
    'gamma': [0, 0.2],
    'colsample_bytree': [0, 0.2, 0.4],
    'max_depth': [3, 4, 5]
}

# initiating our gradient boosted tree
xgb = xgb.XGBClassifier(objective='reg:squarederror', random_state=RANDOM_SEED)
classifier = GridSearchCV(estimator=xgb,
                          param_grid=parameters,
                          cv=5)

# fitting the model
best_xgb = classifier.fit(X_train, y_train)

# view the results of our cross-validated grid search
training_accuracy_report(best_xgb)
```

Code for Boosting Ensemble Model: XGBoost

```
# ProjectPro - https://www.dezyre.com/
# 3.3 - Boosting: AdaBoost
from sklearn.ensemble import AdaBoostClassifier

parameters = {
    'n_estimators': [10, 50, 100],
    'algorithm': ['SAMME', 'SAMME.R'],
    'learning_rate': [0.01, 0.1, 1],
}

# initiating our adaboost classifier
ada = AdaBoostClassifier(base_estimator=None, random_state=RANDOM_SEED)
classifier = GridSearchCV(estimator=ada,
                          param_grid=parameters,
                          cv=5)
# fitting the model
best_ada = classifier.fit(X_train, y_train)

# view the results of our cross-validated grid search
training_accuracy_report(best_ada)
```

Code for Boosting Ensemble Model: AdaBoost

The two popular boosting models XGBoost and AdaBoost, have been explained with code here. More can be found with the attached Google Colab Notebook. We use the `XGBClassifier()` and the `AdaBoostClassifier()` to implement the models.

3.4 Stacking

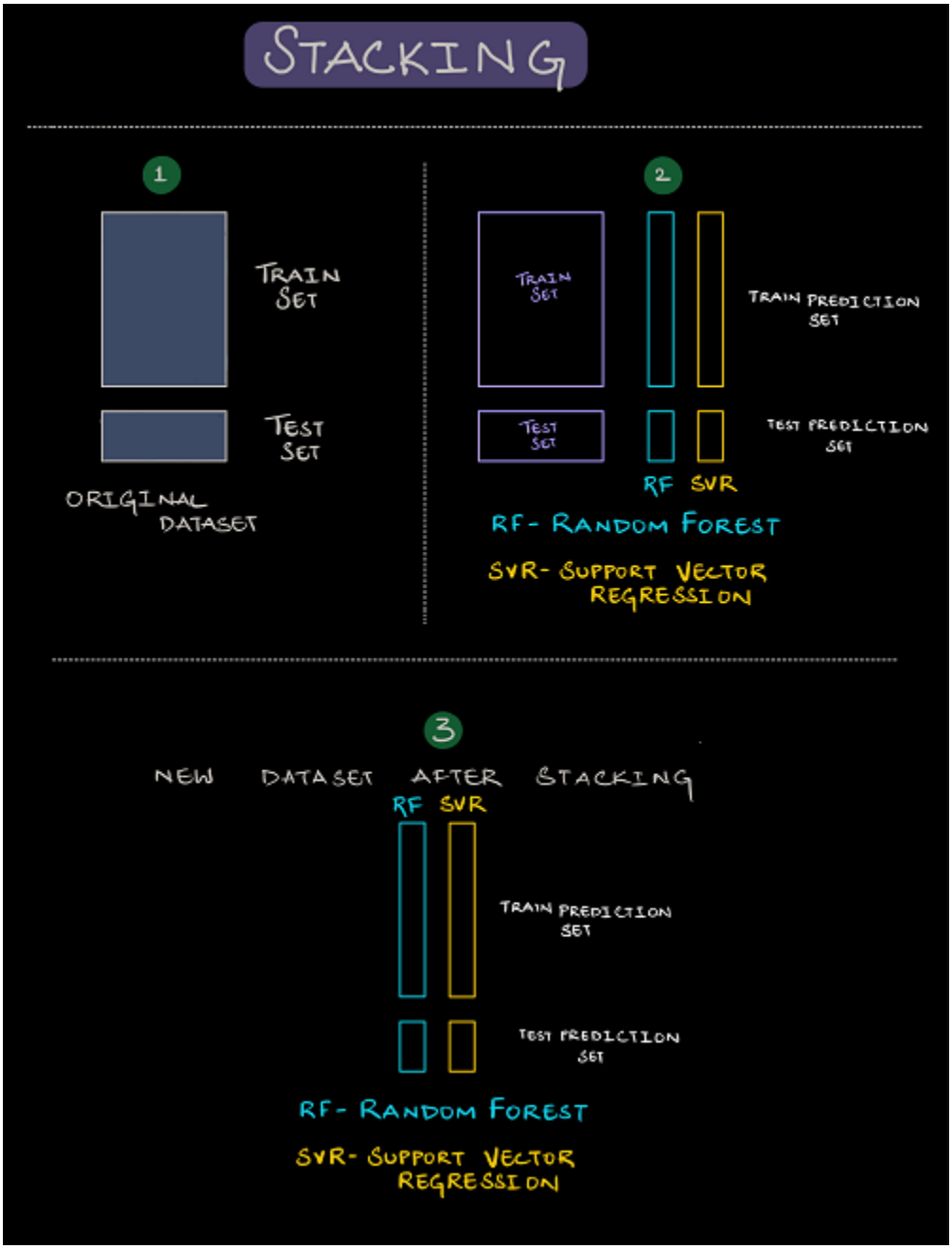
Intuition

Stacking is an advanced model for ensemble learning. I would advise you to read this topic multiple times to understand it completely. The idea of stacking is that we perform predictions on the train and test dataset with a few models. For instance, we run a Random Forest model and get the results. This is done on the train and test dataset. Then, we run a support vector machine algorithm on the train and test data and get the outputs.

Here's the critical part. We now do not consider the original train and test data. Instead, we consider the new decision tree and support vector machine outputs on the train data as the base train model. The new test data is the model outputs of random forest and support vector machine on the test data.

Business Understanding

We will be able to understand more from the figure below. The essence of stacking is that we rely on the derivative models of the base data to make predictions going ahead. Now, if the models were similar, then the outputs would be as well. So, we consciously choose different models to understand the result better, as the models might have better learnt certain parts of the data.



Visual Representation of Stacking

Logic / Code

Stacking considers heterogeneous weak learners. Stacking combines several weak learners and combines them by training a meta-model to output predictions based on multiple predictions returned by these weak learners.

```
# ProjectPro - https://www.dezyre.com/
# 3.4 - Stacking
# Choosing Random Forest and Support Vector Machine as the classifiers
estimators = [
    ('rf', RandomForestClassifier(n_estimators=10, random_state=42)),
    ('svr', make_pipeline(StandardScaler(),
                          LinearSVC(random_state=42)))
]

# Implementing Stacking
clf = StackingClassifier(
    estimators=estimators, final_estimator=LogisticRegression()
)

# Train-Test-Split with stratifying
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, stratify=y, random_state=RANDOM_SEED
)
```

```
# Final Fit
resultStacking = clf.fit(X_train, y_train).score(X_test, y_test)
print(f'Stacking Output: {resultStacking*100:,.2f}%')
```

```
Stacking Output: 80.63%
```

Code for Stacking Ensemble

Here, we use the `StackingClassifier()` from sklearn to implement stacking of random forest and support vector machine. This highlights the examples as explained in the Intuition and Business Understanding Sections. For more, please feel free to have a look at the Google Colab Notebook.

3.5 Blending

Intuition

Blending follows a similar approach to stacking. The only difference is that in Blending, a holdout validation set is leveraged to make predictions. Predictions in the validation set will be used to train the meta-model. Forecasts in the test set will be used to test the meta-model.

Business Understanding

Predictions of the validation set become the training data of the meta-model. Forecasts of the test set will become the test data of the meta-model. These predictions are stacked and used for the final model.

Logic / Code

The sklearn library does not natively support Blending. So, we will leverage a custom code to implement what we have discussed so far.

```
# ProjectPro - https://www.dezyre.com/
# 3.4 - Stacking
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=TEST_RATIO, random_state=1)
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=TEST_RATIO, random_state=1)
```

```
# Model 1 is a Support Vector Machine
model1 = SVC(gamma='scale', probability=True, random_state=RANDOM_SEED)
model1.fit(x_train, y_train)
val_pred1=model1.predict(x_val)
test_pred1=model1.predict(x_test)
```

```
# Model 2 is a Random Forest Classi
model2 = RandomForestClassifier(random_state=RANDOM_SEED)
model2.fit(x_train,y_train)
val_pred2=model2.predict(x_val)
test_pred2=model2.predict(x_test)
```

```
Combining the meta-features and the validation set, a logistic regression model is built to make predictions on the test set.
```

+ Code + Text

```
# Blending!
df_val=pd.concat([x_val, val_pred1,val_pred2],axis=1)
df_test=pd.concat([x_test, test_pred1,test_pred2],axis=1)

model = LogisticRegression()
model.fit(df_val,y_val)
resultBlending = model.score(df_test,y_test)
print(f'Blending Output: {resultBlending*100:,.2f}%')
```

```
Blending Output: 79.21%
```

Code for Blending Ensemble

This code for ensemble models using Blending. The train test and validation set are leveraged with a Logistic Regression model in this example above.

Can Ensemble Learning be used for both regression and classification problems?

The short answer

Yes, ensemble learning can be used for both regression and classification problems.

The detailed answer

Specific ensemble learning methods such as max voting work for classification problems. Techniques such as Gradient Boosting (GBM) work for both classification and regression problems. In the examples we have stated above, multiple instances of regression and classification have been mentioned. The ensemble models work by reducing bias and variance to enhance the accuracy of models**.**

5. How do these ensemble learning techniques help improve the performance of the machine learning model?

The short answer

Ensemble Learning helps improve machine learning results by combining the learning and methodology compared to a single model. This is highlighted in our example in the Introduction Section of the blog.

The detailed answer

Ensemble models are meta algorithms that combine several machine learning techniques into a single predictive ensemble model to decrease variance (Bagging), bias (boosting) or improve predictions (stacking). There are two groups of ensemble models when viewed from the lens of performance types:

- *Sequential* ensemble models – The logic employed is to leverage the dependence between the base learners. Thus, the mistakes made by the first model are sequentially corrected by the second model and so on. This helps get the most accurate ensemble possible. For example, the AdaBoost Ensemble Model is sequential

- *Parallel* ensemble models – The logic employed is to leverage the independence between the base learners. Thus, the mistakes in labelling made by one model are different from those from another independent model. This lets the ensemble model average out the errors.
  For example, the Random Forest Model is a parallel ensemble model with independent Decision Trees

An interesting point to note here is that combining stable learners is less advantageous since the ensemble will not help improve generalisation performance.

6 Key Takeaways

If you have spent time and gone through the entire blog and the code on the Google Colab Notebook, you have learnt the theory and implementation of Ensemble Models.

If you would like the best of real-world Data Science Projects, ProjectPro is the platform to work on end to end, along with mini-videos to explain core concepts and learning by doing. To recap, the key takeaways from this long but fun blog on Ensemble models are:

- Some of the best models that have been implemented in multiple use cases are ensemble models. Therefore, knowing how ensemble models work can starkly improve model performance.
- Ensemble Learning is the process wherein multiple models (weak learners) are combined to make a single integrated model (strong learner)

- Fundamental ensemble models can give great generalised results without increasing complexity, and these are useful in machine learning used in production (MLOps)
- Complex ensemble models such as bagging and boosting tend to yield the best performance, but at the cost of increased complexity and computing power
- Ensemble models can be used for a plethora of problem types. They can be used for classification and regression problems
- Ensemble models can be divided into two broad groups of sequential ensemble models and parallel ensemble models
- The best way to learn Data Science is to DO Data Science!

Congratulations! You have managed to get so far all that way from the start. You are now ready to take on a new dataset and attempt to create an ensemble model of your own.

Author

Anish Mahapatra
Senior Data Scientist
LinkedIn: https://www.linkedin.com/in/anishmahapatra (LinkedIn)