**Blending Ensemble Learning**

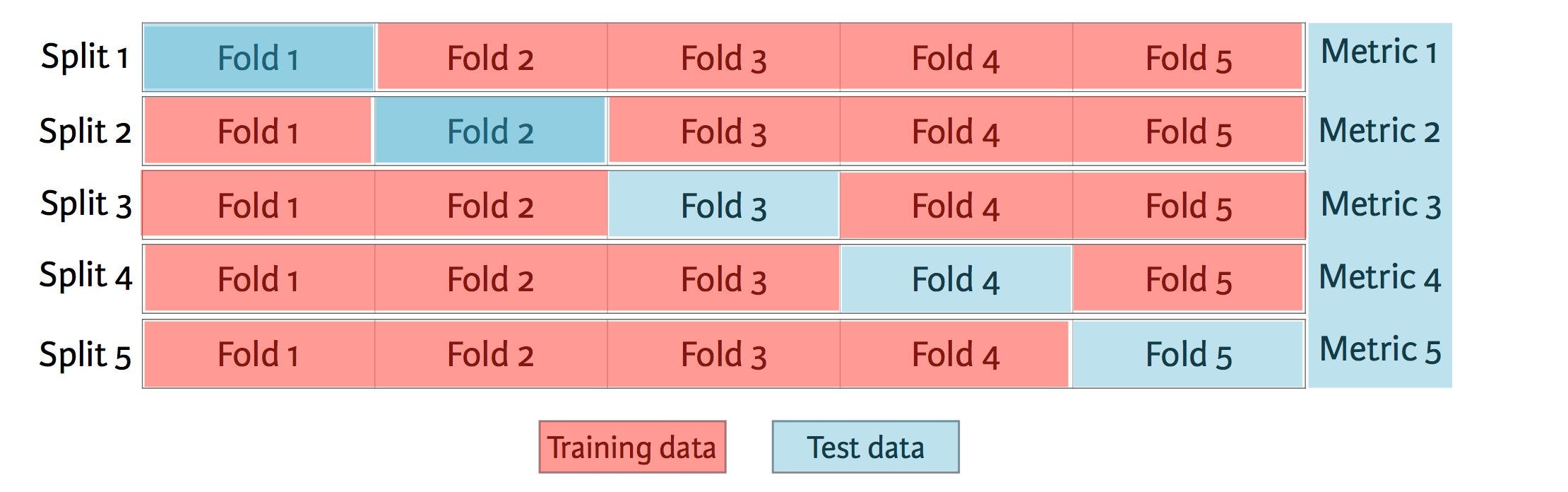
**Hold-out**

Hold-out is when you split up your dataset into a ‘train’ and ‘test’ set. The training set is what the model is trained on, and the test set is used to see how well that model performs on unseen data. A common split when using the hold-out method is using 80% of data for training and the remaining 20% of the data for testing.

**Cross-validation**

Cross-validation or ‘k-fold cross-validation’ is when the dataset is randomly split up into ‘k’ groups. One of the groups is used as the test set and the rest are used as the training set. The model is trained on the training set and scored on the test set. Then the process is repeated until each unique group as been used as the test set.

For example, for 5-fold cross validation, the dataset would be split into 5 groups, and the model would be trained and tested 5 separate times so each group would get a chance to be the test set. This can be seen in the graph below.



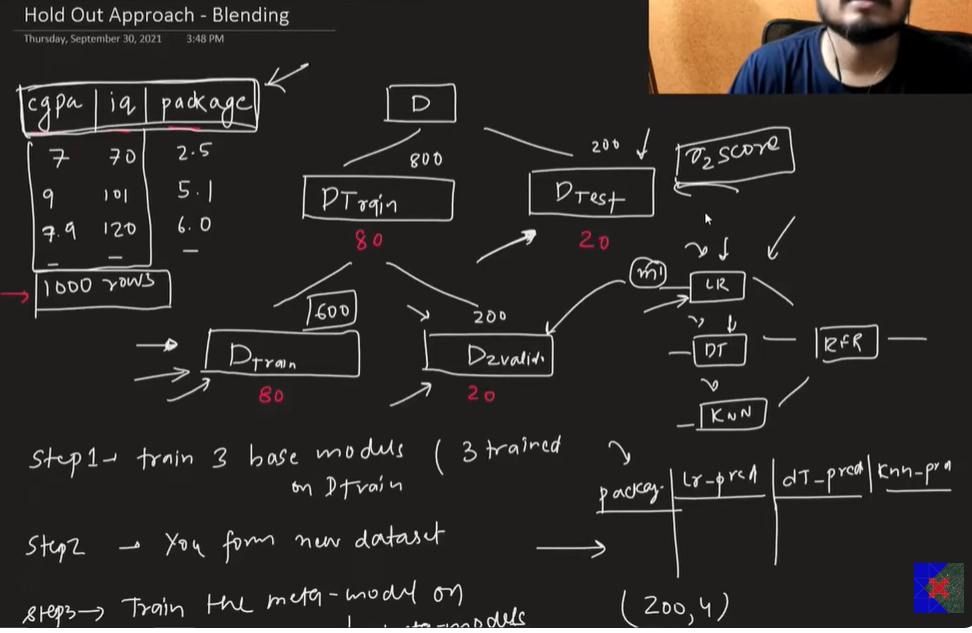
5-fold cross validation ([image credit](https://www.datacamp.com/))

**Hold-out vs. Cross-validation**

Cross-validation is usually the preferred method because it gives your model the opportunity to train on multiple train-test splits. This gives you a better indication of how well your model will perform on unseen data. Hold-out, on the other hand, is dependent on just one train-test split. That makes the hold-out method score dependent on how the data is split into train and test sets.

The hold-out method is good to use when you have a very large dataset, you’re on a time crunch, or you are starting to build an initial model in your data science project. Keep in mind that because cross-validation uses multiple train-test splits, it takes more computational power and time to run than using the holdout method.

To see an example of comparing hold-out and cross-validation while testing a machine learning model, check out my post [here](https://towardsdatascience.com/building-a-k-nearest-neighbors-k-nn-model-with-scikit-learn-51209555453a). The article shows how testing a model varies between both methods and has uses the python Scikit-learn library to implement both methods.



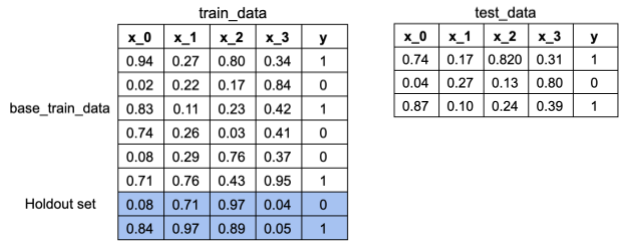
**Blending (Hold out Method)**

In blending, the concept of holdout set is introduced which means that out of 8 records lets say 2 records will be used as holdout set. So each model will learn on the remaining 6 records and then give prediction on this holdout set containing 2 records. And then these predictions along with actual output of these records, will be sent as the dataset for the meta model, and once this model learns or trains from the dataset (where the input values are predictions of different inputs and output is the actual value), this meta model will be then used to give predictions on the test dataset

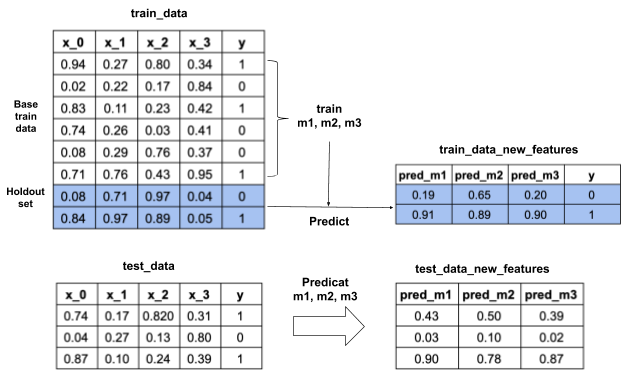
**Key Characteristics of Blending:**

* **Training Data**: The original dataset is split into two parts:
* **Training Set**: Used to train the base models.
* **Holdout Set**: Used to generate predictions from the base models which are then used to train the meta-model.
* **Meta-Model**: The meta-model is trained on the predictions made by the base models on the holdout set. This meta-model learns to combine the base models’ predictions to make the final prediction.

**Step 1**: train\_data is split into base\_train\_data and holdout\_set.



**Step 2**: Base models are fitted on base\_train\_data, and predictions are made on holdout\_set and test\_data. These will create new prediction features.



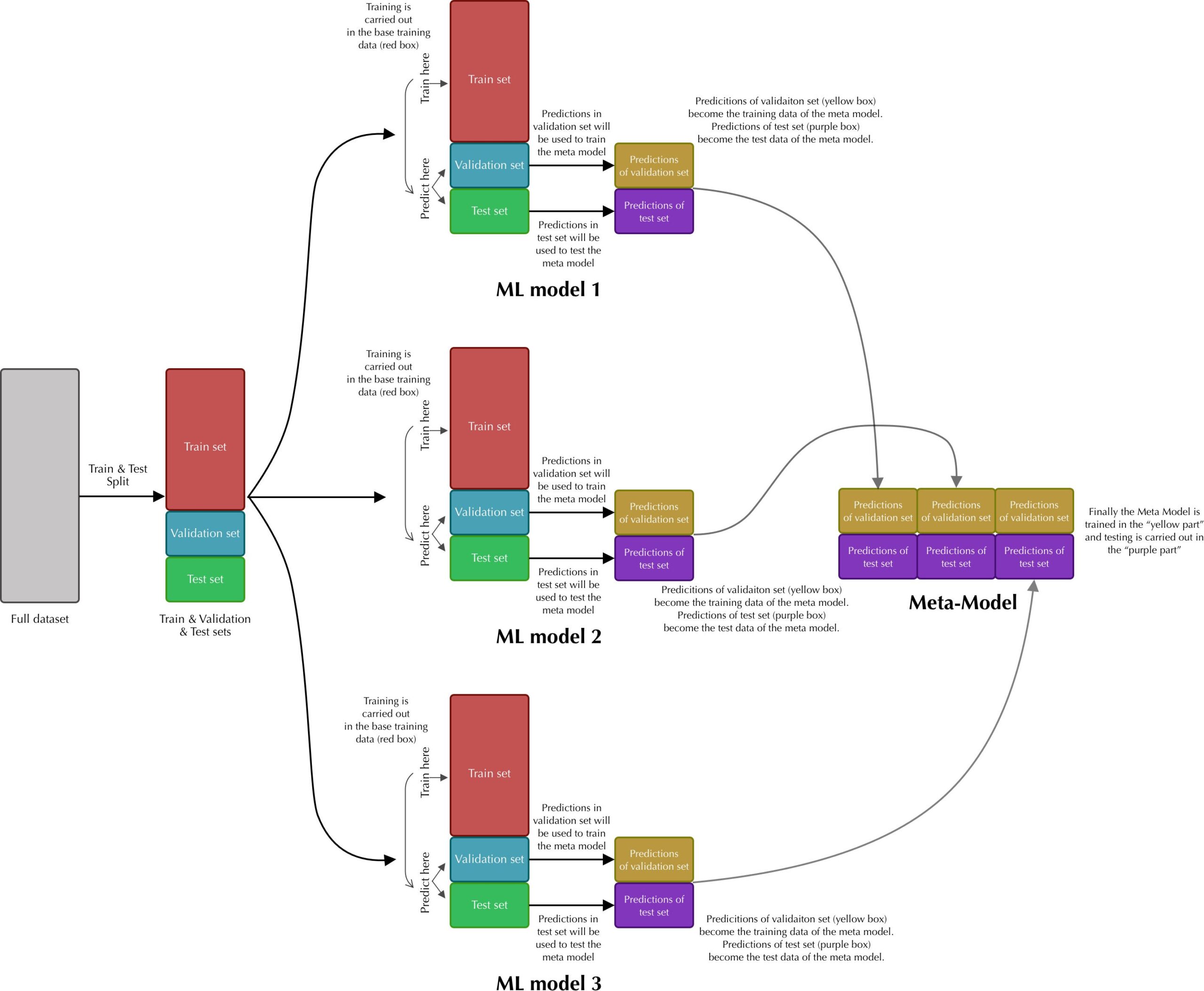
**Step 3**: A new meta-model is then fit on holdout\_set with new prediction features. Both original and meta features from holdout\_set will be used.

**Step 4**: The trained meta-model is used to make final predictions on the test data using both original and new meta features.

**Blending**

**Blending** is a technique derived from **Stacking Generalization**. The only difference is that in **Blending**, the *k-fold cross validation* technique is not used to generate the training data of the *meta-model.* Blending implements "*one-holdout set*", that is, a small portion of the training data (*validation*) to make predictions which will be "*stacked*" to form the training data of the *meta-model*. Also, predictions are made from the test data to form the *meta-model* test data.

In figure 3 we can see a **Blending** architecture using 3 base models (*weak learners*) and a *final classifier*. The *blue boxes* represent that portion of the training data that is used to generate predictions (yellow boxes) to form the *meta-model*. The *green boxes* represent the test data which is used to generate predictions to form the *meta-model* test data (purple boxes).

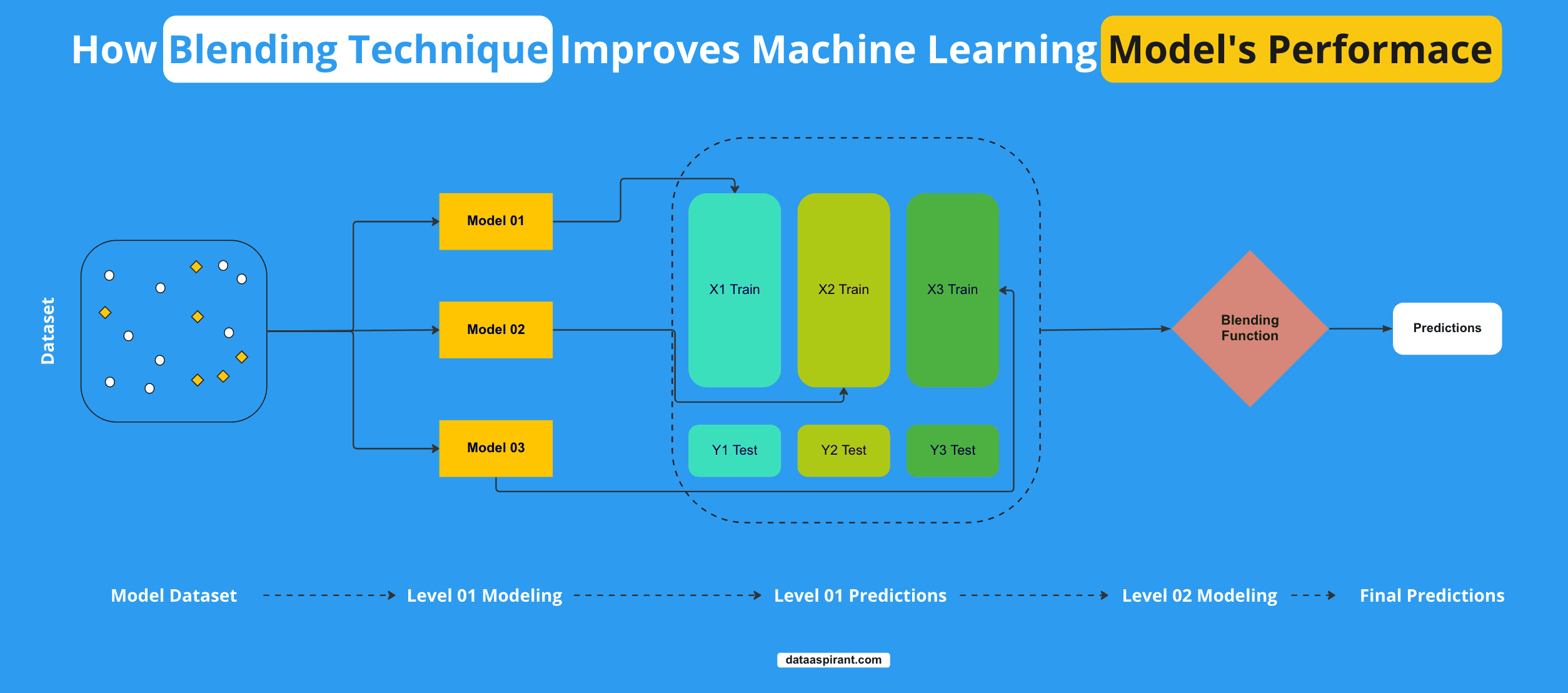
Figure 3. Blending Architecture| Image by author

Great, now that you’re familiar with the **Blending** architecture,

**How Blending Technique Algorithm Works**

Four types of ensemble methods are used for real-time model training and predictions:

* Voting ensembles,
* Bagging techniques,
* Boosting techniques,
* Stacking ensembles.



Blending is the **variations of**[**stacking method**](https://dataaspirant.com/stacking-technique/), where multiple layers of algorithms are used to train and predict the data.

Unlike other ensemble methods, which use multiple algorithms in **single layers**, stacking and blending use a number of layers with various algorithms.

Here mainly, two levels/layers are used.

In the first layer, we have multiple algorithms called **base models,** and in the second or last layer, we have one **deciding algorithm** called **metamodel.**

The general approach in stacking and blending algorithms is that the data is first fed to the base model for the training. Then the outputs from the base model are provided to the metamodel as a training set, and once the metamodel is trained, the result is considered the final result of the algorithms.

If we specifically talk about the blending approach, the dataset is fed to the base models first. The algorithms in base models get trained on the training data provided, and once they are trained, the test set is given to them for the prediction of the same.

The base models are now trained to predict test data, and whatever results they predict for the test data will be fed to the metamodel or the algorithms in the last player for training.

Now the **metamodel** will be trained on fed data, and the results or accuracies that the metamodel will return will be considered the final results of the algorithm.

One important thing to note is that the stacking and blending working mechanism is almost identical. Just the stacking algorithms split the data using K parts, and then the base models are trained, whereas the blending approach classically splits the data in a conventional way, and then the base models are trained on the same.

**Challenges With Blending Technique**

In previous sections we discussed that we first divide the dataset into parts, training and testing set, and then the training set is provided to the base models for the training purpose.

Once the base models are trained, the test set is provided, and the results of the test set are used as a training set of the metamodel. The problem lies here, where we train the **metamodel** with test results from the base models.

As we are training our metamodel on already known data, the results obtained from the base models on the test set as base models have already seen the dataset.

Here we can clearly understand that the data is leakage, which can cause poor performance of the model, which may not give reliable results.

For example, let us say we have a dataset of 100 rows, and we split the same into training sets of 70 and testing sets of 30. Now we will train the base model with the training set of 70 rows, and once the model is trained, now it is ready for the prediction phase.

Now to train the metamodel, we will need training data for the same. To get the same, we simply perform predictions on the base model on the test data set of 30 rows.

The base model will be trained on the same, and whatever results from the base model gives as an output will be used as the training set for the meta models.

Here we can see that we used the test set from the prediction on base models and we used the results of the same for  the training of the meta-model; here, we can see the case of **data leakage**, and this needs to be resolved to get a reliable and accurate model.

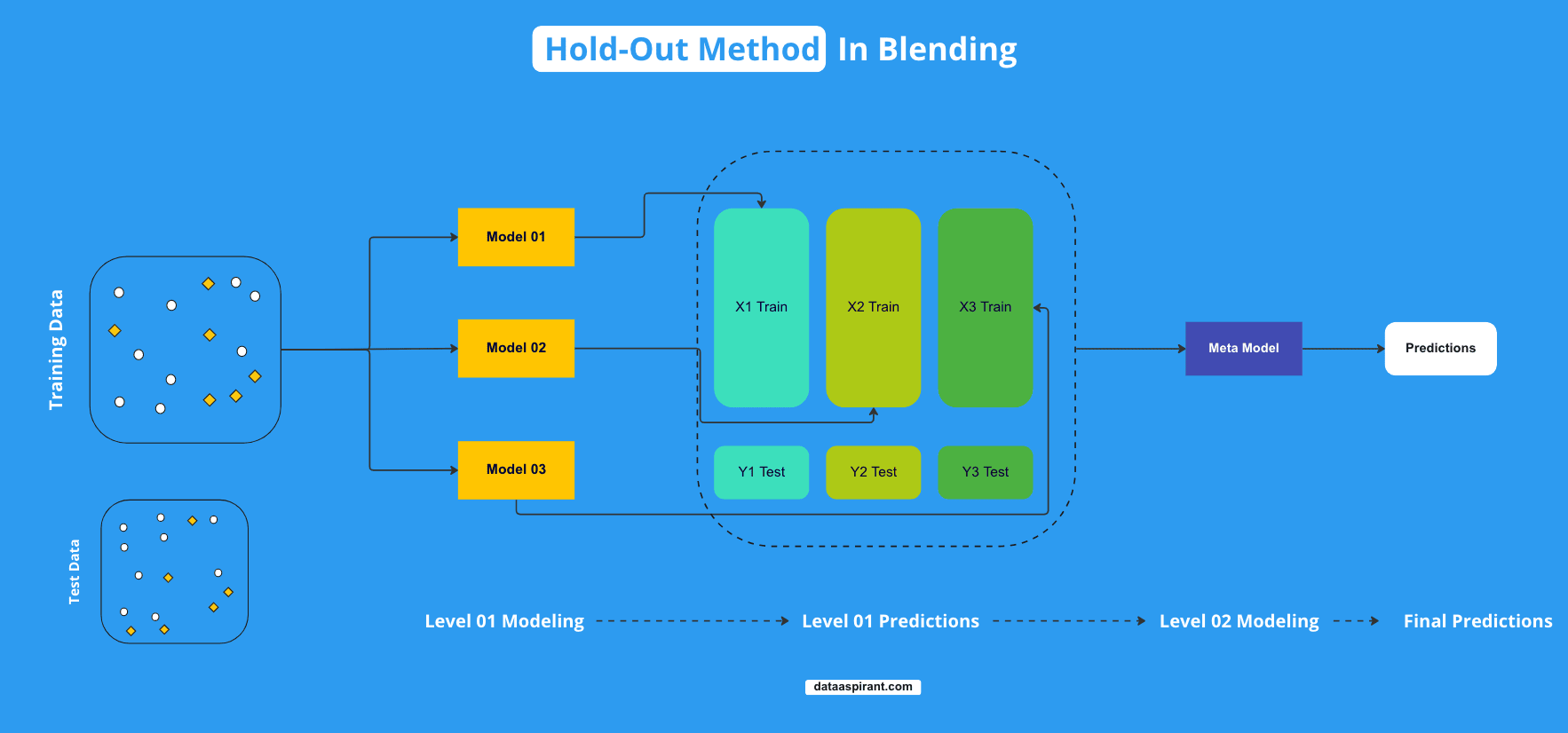
Now to overcome these issues, there can be only one solution.

We need to stop data leakage somehow.

**Hold-Out Method in Blending**

Previously, we saw that we were using the test set for the prediction on the base model. Then the results from the same are used for training of the meta-model, which is somehow leading to the problem of data leakage and which needs to be resolved.

To overcome this issue, a method called the **hold-out approach** is used in blending models.



In the hold-out method of the blending algorithms, we change the way of splitting the data into training and testing sets.

Until now, we have been splitting the data into two parts, training and testing sets, but from now on, we will split the data into **three parts**: training, testing, and validation split.

So here, first, the whole dataset will be split into two parts: training and testing datasets. Then the training dataset will again be sp, fitted into two parts: training and validation datasets.

Now the problem of data leakage is solved, as the training dataset will be used to train the base models. The validation data will be used as a test dataset for base models.

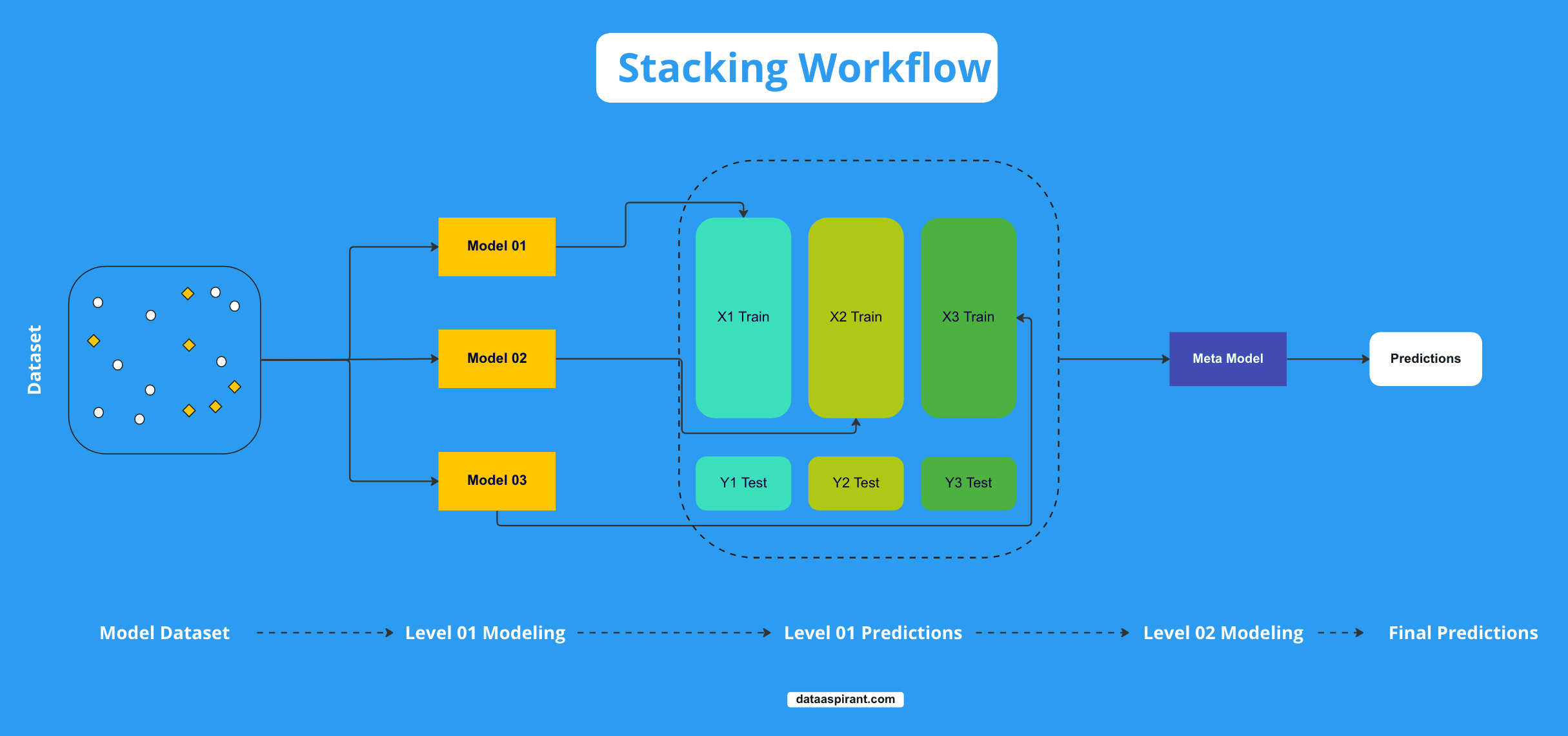
Now whatever the base models predict results will be given to the metamodel for training, and the actual; test model from the first split of the data will be used to evaluate the final model that is obtained after the training of the metamodel.

**Difference Between Stacking and Blending**

Stacking and blending are two different ensemble learning techniques used to combine the predictions of multiple base models (also called base learners) to improve overall predictive performance.

Let me explain each technique and its workflow.

**Stacking Workflow**

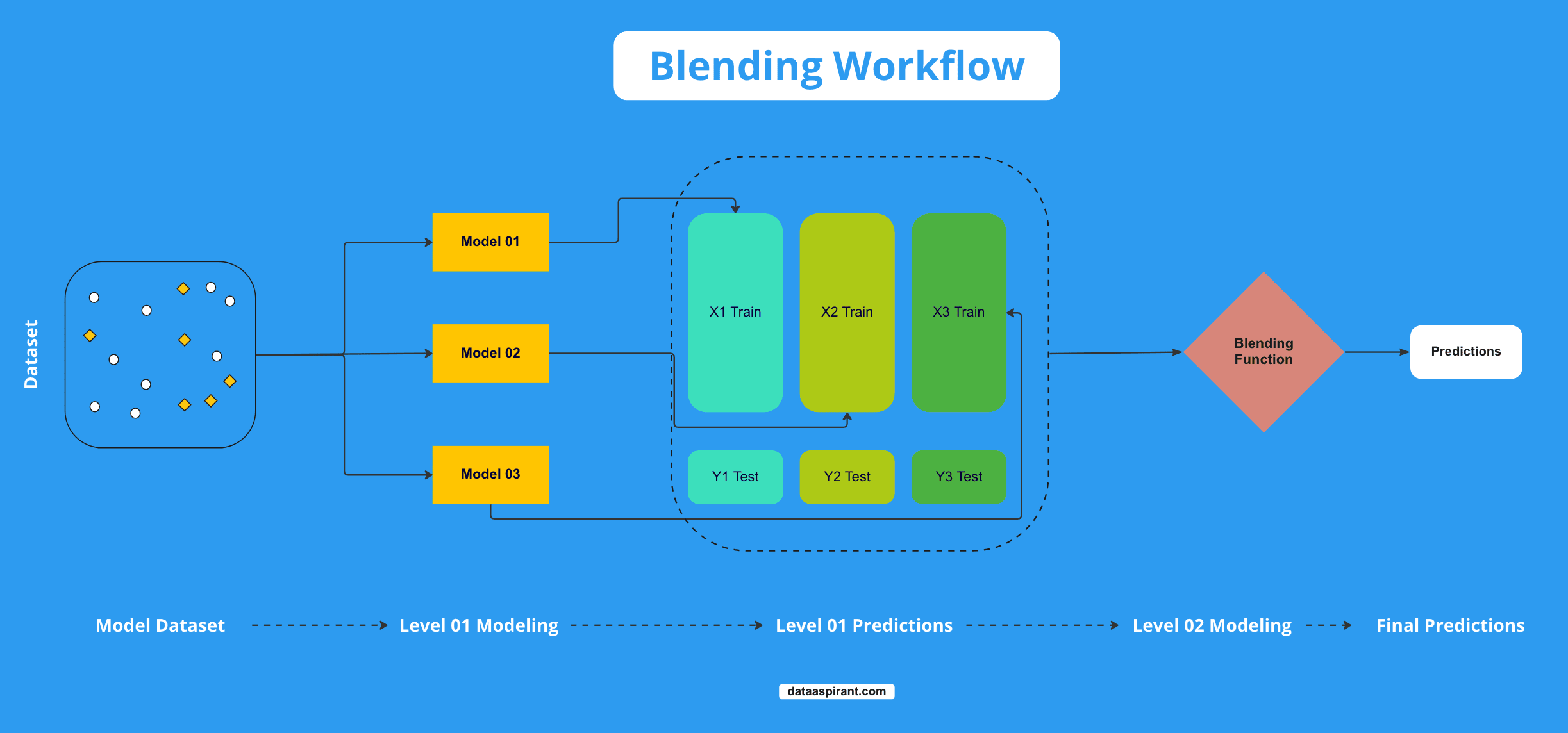


Stacking, also known as [**stacked generalization**](https://dataaspirant.com/stacking-technique/), involves training a meta-model (also called a second-level model) on the predictions of multiple base models. The workflow for stacking is as follows:

1. Split the training data into **K-folds** (for example, using K-fold cross-validation).
2. Train each base model on **K-1 folds** and make predictions on the remaining fold.
3. Repeat this process for all **K folds,** resulting in a set of out-of-fold predictions for each base model.
4. Combine these out-of-fold predictions to create a new dataset, which serves as the input features for the **meta-model.**
5. Train the meta-model on this new dataset, with the original target variable as the output.
6. To make predictions on new, unseen data, feed the data through the base models to generate predictions, and then use these predictions as input features for the meta-model.

In stacking, the focus is on leveraging the strengths of different base models by learning how to optimally combine their predictions through the meta-model.

**Blending Workflow**



Blending is a simpler technique compared to stacking. It involves training base models independently and then combining their predictions using a weighted average, majority vote, or another aggregation method. The workflow for blending is as follows:

1. Split the training data into two parts: a **training set and a validation set.**
2. Train each base model on the training set.
3. Make predictions using the base models on the validation set and new, unseen data.
4. Combine these predictions using a predefined blending function an aggregation method (e.g., weighted average, majority vote) to produce the final output.

The main difference between stacking and blending lies in the way the base models' predictions are combined. Stacking uses a meta-model to learn the optimal combination, whereas blending relies on a predefined aggregation method.

In summary, stacking is a more complex ensemble technique that trains a meta-model to learn the optimal way of combining base model predictions, whereas blending is a simpler approach that involves aggregating base model predictions using a predefined method.

Both techniques aim to improve [**overall predictive performance**](https://dataaspirant.com/six-popular-classification-evaluation-metrics-in-machine-learning/) by leveraging the strengths of multiple base models.

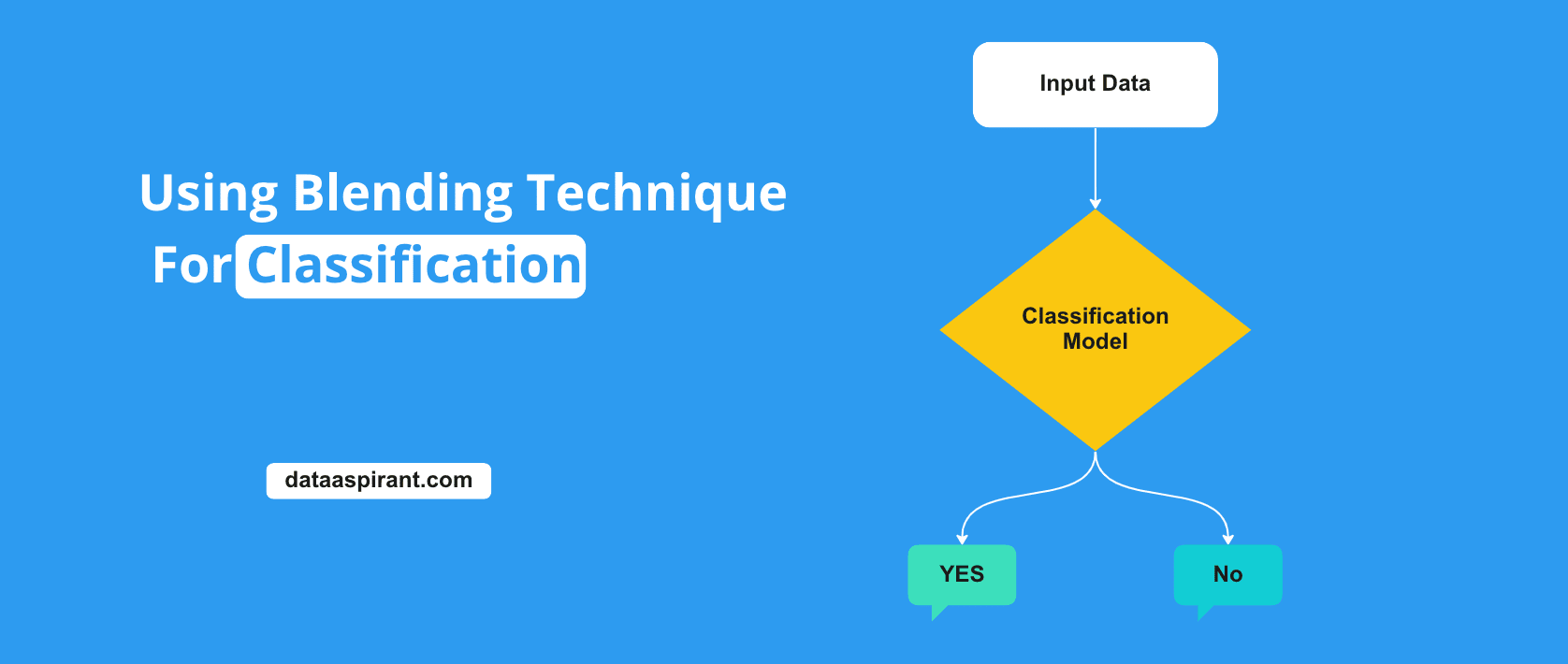
**Using Blending Algorithm for Both Classification and Regression Problems**

As we discussed above, blending can be used for both [**regression and classification problems**](https://dataaspirant.com/classification-and-prediction/); the exact mechanisms follow in both approaches.

**How to Use Blending Technique for Classification**

As we know that in classification problems, the output or target column is textual or has categories or labels to classify.

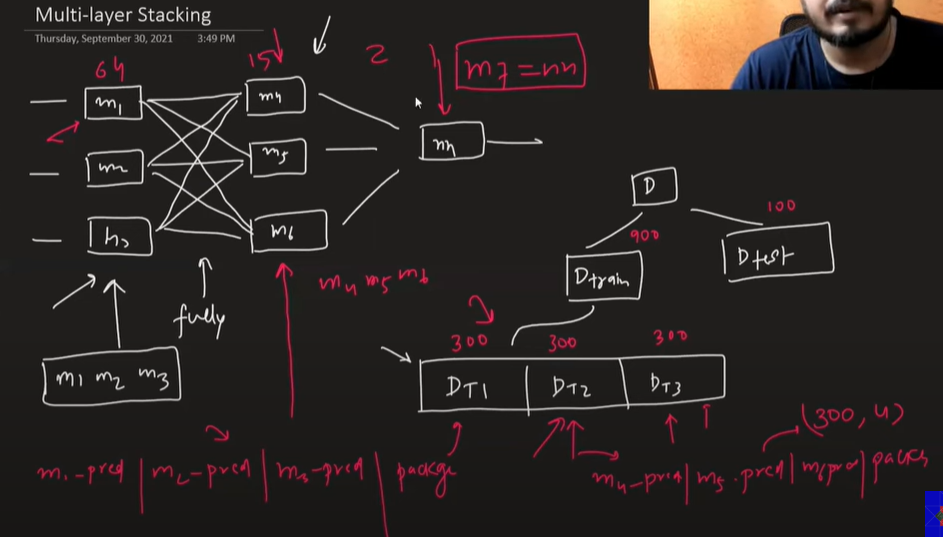
As per the working of the blending algorithm, we first feed the training data that is split from the whole dataset to the model, and the base models will be trained on the same.



So here, basically, the base model will be fed with the training data, and each of the classifiers or the model will be trained on the same. Here the algorithms or the classifiers can be any algorithm like [**logistic regression**](https://dataaspirant.com/how-logistic-regression-model-works/), [**decision trees**](https://dataaspirant.com/how-decision-tree-algorithm-works/), etc.

Once the base models are trained, the prediction are taken from the base models, and the final output is decided on the basis of either **majority voting** or the **weighted average**of all the classifiers where the weights to the outputs are assigned.

**Multi-Layer stacking:**



**Blending: Code Example**

To apply the blending algorithm, one will need to write the code manually, as there is not any built-in module created for this algorithm.

To apply blending on any dataset, use the code below:

**Importing Required Libraries:**

**from** numpy **import** hstack

**from** sklearn.datasets **import** make\_classification

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.metrics **import** accuracy\_score

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.neighbors **import** KNeighborsClassifier

**from** sklearn.tree **import** DecisionTreeClassifier

**from** sklearn.svm **import** SVC

**from** sklearn.naive\_bayes **import** GaussianNB

**Creating base Models List:**

**def** **get\_models**():

models = list()

models.append(('lr', LogisticRegression()))

models.append(('knn', KNeighborsClassifier()))

models.append(('cart', DecisionTreeClassifier()))

models.append(('svm', SVC()))

models.append(('bayes', GaussianNB()))

**return** models

**Fit the ensembles:**

**def** **fit\_ensemble**(models, X\_train, X\_val, y\_train, y\_val):

meta\_X = list()

**for** name, model **in** models:

# fit in training set

model.fit(X\_train, y\_train)

# predict on hold out set

yhat = model.predict(X\_val)

# reshape predictions into a matrix with one column

yhat = yhat.reshape(len(yhat), 1)

# store predictions as input for blending

meta\_X.append(yhat)

# create 2d array from predictions, each set is an input feature

meta\_X = hstack(meta\_X)

# define blending model

blender = LogisticRegression()

# fit on predictions from base models

blender.fit(meta\_X, y\_val)

**return** blender

**def** **predict\_ensemble**(models, blender, X\_test):

# make predictions with base models

meta\_X = list()

**for** name, model **in** models:

# predict with base model

yhat = model.predict(X\_test)

yhat = yhat.reshape(len(yhat), 1)

# store prediction

meta\_X.append(yhat)

# create 2d array from predictions, each set is an input feature

meta\_X = hstack(meta\_X)

# predict

**return** blender.predict(meta\_X)

# create the base models

models = get\_models()

# train the blending ensemble

blender = fit\_ensemble(models, X\_train, X\_val, y\_train, y\_val)

# make predictions on test set

yhat = predict\_ensemble(models, blender, X\_test)

# evaluate predictions

score = accuracy\_score(y\_test, yhat)

print('Blending Accuracy: %.3f' % (score\*100))

code –

<https://www.scaler.com/topics/machine-learning/blending-in-machine-learning/>

**4. Voting (Simple Voting)**

In **voting**, multiple models are trained independently on the same dataset, and their predictions are combined by **voting** in the case of classification tasks, or by **averaging** in the case of regression tasks. This is one of the simplest ensemble methods and can be classified into two types: **hard voting** and **soft voting**.

* **Hard Voting**: In classification tasks, the final ensemble prediction is determined by selecting the class that receives the most votes from the base models’ predictions. This is often referred to as “hard voting.”
* **Soft Voting**: In regression tasks, the final prediction is typically obtained by averaging the predictions of the base models. This is also known as “soft voting.”

**Example:**

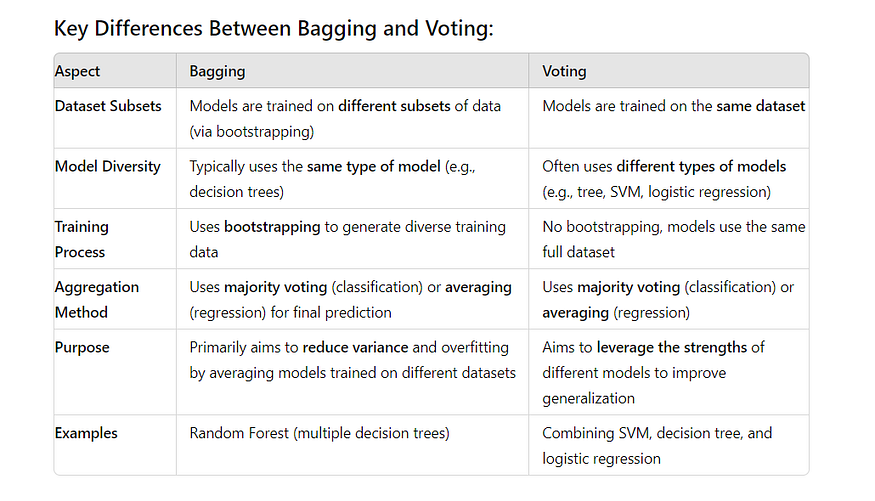
You can train three models (e.g., logistic regression, decision tree, and random forest) on a dataset and combine their predictions by hard voting. The final prediction is based on the majority vote.

**Advantages of Voting:**

* Simple to implement and interpret.
* Can improve accuracy by combining diverse models.
* Works well when the base models are fairly strong and complementary.

**Difference between bagging and voting**

Press enter or click to view image in full size



**1. What is Stacking in machine learning?**

**Answer:**  
Stacking (Stacked Generalization) is an **ensemble learning technique** that combines multiple base models (level-0 models) through a meta-model (level-1 model).

* The base models learn from the original dataset.
* Their predictions are used as input features for the meta-model.
* The meta-model learns to combine base model outputs for better accuracy.

**2. How does Stacking work step-by-step?**

**Answer:**  
Let’s assume we have **1200 records**:

1. **Split training data into K folds** (e.g., K=5).
2. **Base models training**:
   * For each fold:
     + Train the base model on K-1 folds.
     + Predict on the remaining fold.
     + Store predictions.
3. **Out-of-fold predictions**:
   * After K iterations, you’ll have predictions for all rows without data leakage.
4. **Meta-model training**:
   * These predictions become new features.
   * Train the meta-model on them using the true labels.
5. **Final prediction**:
   * Base models are trained on the **full dataset**.
   * Their predictions for test data go into the meta-model to produce the final prediction.

**3. Why do we need out-of-fold predictions in stacking?**

**Answer:**  
To prevent **data leakage**. If we trained base models and generated predictions on the same data they learned from, the meta-model would get overly optimistic signals, leading to overfitting.

**4. What’s the difference between Stacking and Bagging?**

**Answer:**

* **Bagging**:
  + Uses **same type of model** (e.g., Random Forest = multiple decision trees).
  + Aggregates predictions by averaging or voting.
  + Focuses on **reducing variance**.
* **Stacking**:
  + Uses **different model types**.
  + Combines predictions through a meta-learner.
  + Focuses on **reducing both bias & variance**.

**5. What’s the difference between Stacking and Blending?**

**Answer:**

| **Feature** | **Stacking** | **Blending** |
| --- | --- | --- |
| **Training data split** | Uses **K-fold cross-validation** | Uses a **hold-out validation set** |
| **Meta-model input** | Out-of-fold predictions | Predictions from hold-out set |
| **Leakage control** | Strong (CV ensures no leakage) | Weaker (hold-out might not generalize well) |
| **Complexity** | Higher | Lower |
| **Data usage** | Uses almost all training data | Wastes part of data for hold-out |

**6. When would you prefer Blending over Stacking?**

**Answer:**

* When **speed** is more important than absolute accuracy.
* When you have **very large datasets**, where holding out 10–20% still leaves enough data for training.
* When you want a **simpler implementation** without K-fold complexity.

**7. Can you overfit in stacking? How to avoid it?**

**Answer:**  
Yes, if:

* Meta-model is too complex.
* Base model predictions are highly correlated.

**Avoidance strategies:**

* Use **regularization** in meta-model (e.g., Lasso, Ridge).
* Choose **diverse base models**.
* Limit the depth or complexity of base models.
* Use **cross-validation** correctly.

**8. What types of models can be used as meta-learners?**

**Answer:**  
Any supervised model:

* **Linear models** (Linear Regression, Logistic Regression) – for interpretability.
* **Tree-based models** (LightGBM, XGBoost) – for non-linear relationships.
* **Neural networks** – for highly complex combinations.

**9. Can you stack classifiers with regressors?**

**Answer:**  
Yes, but:

* All base models and meta-model must match the **prediction type** (classification or regression).
* You cannot directly mix classifiers and regressors unless you transform outputs appropriately.

**10. How do you choose base models in stacking?**

**Answer:**

* Pick models with **different biases** (e.g., linear + tree-based + SVM).
* Avoid using models that are too similar (e.g., three gradient boosting models with the same parameters).
* Test correlation between base model predictions; lower correlation often means better stacking gains.

**11. What is "level-1 feature space" in stacking?**

**Answer:**  
It’s the **dataset of predictions** generated by base models that is used to train the meta-model.

* If you have 3 base models, level-1 feature space will have 3 columns (one per base model prediction).

**12. Example: How many records does meta-model get in stacking?**

**Answer:**  
If you have **1200 records**:

* After K-fold CV (say K=5), meta-model still gets **1200 rows** in training, but **features** are base model predictions instead of original features.