**Stacking Ensemble Learning**

<https://www.kaggle.com/code/dhanishahahaha/stacking-and-blending-ensemble-techniques>

**What is Stacking (Stacked Generalization)**

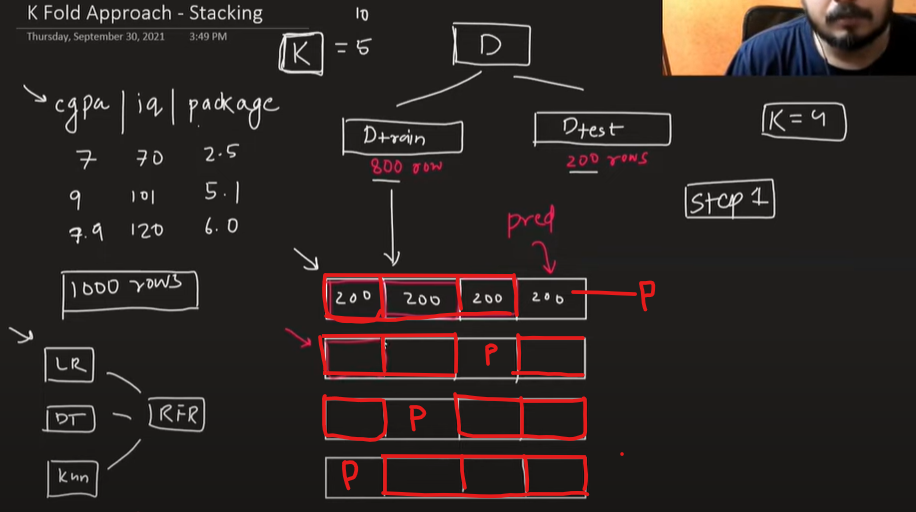
Stacking, also known as Stacked Generalization is an ensemble technique that combines multiple classifications or regression models via a meta-classifier or a meta-regressor. The base-level models are trained on a complete training set, then the meta-model is trained on the features that are outputs of the base-level model. The base-level often consists of different learning algorithms and therefore stacking ensembles are often heterogeneous. Here is a diagram illustrating the process

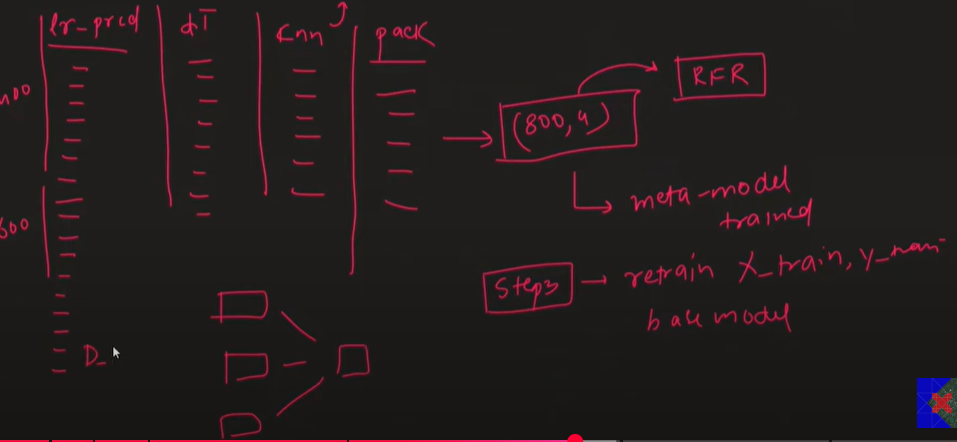
The models(Base-Model) in stacking are typically different (e.g. not all decision trees) and fit on the same dataset. Also, a single model( Meta-model) is used to learn how to best combine the predictions from the contributing models

The architecture of a stacking model involves two or more base models, often referred to as level-0 models and a meta-model. Meta-model, also referred to as a level-1 model combines the predictions of the base models

The predictions made by base models on out-of-sample data is used to train meta-model. We can understand the process in the following steps

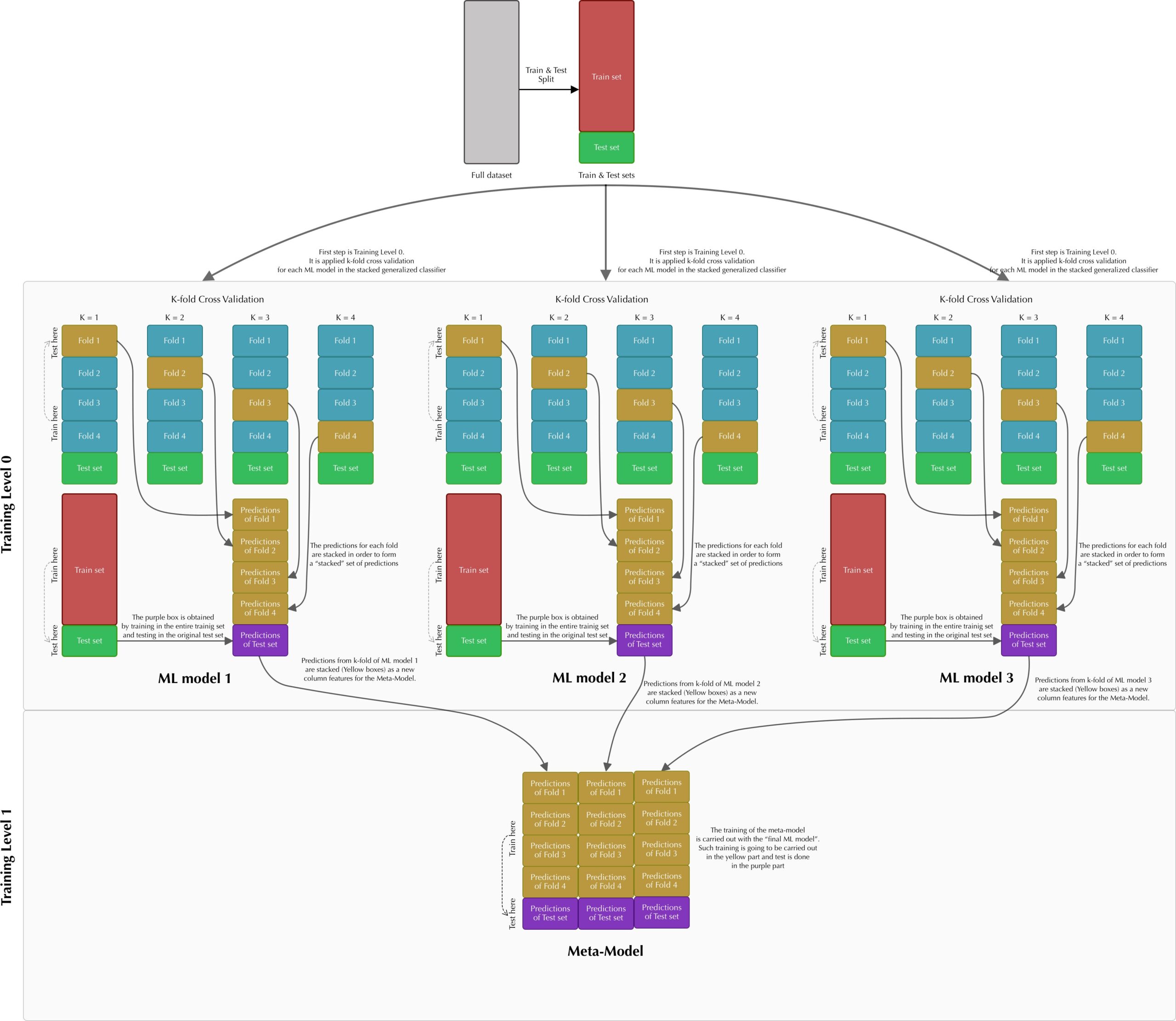
1. We split the data into two parts viz, a training set and test set. The training data is further split into K-folds just like K-fold cross-validation.
2. A base model(e.g k-NN) is fitted on the K-1 parts and predictions are made for the Kth part.
3. This process is iterated until every fold has been predicted.
4. The base model is then fitted on the whole train data set to calculate its performance on the test set.
5. We repeat the last 3 steps for other base models.(e.g SVM,decision tree,neural network etc )
6. Predictions from the train set are used as features for the second level model.
7. Second level model is used to make a prediction on the test set.

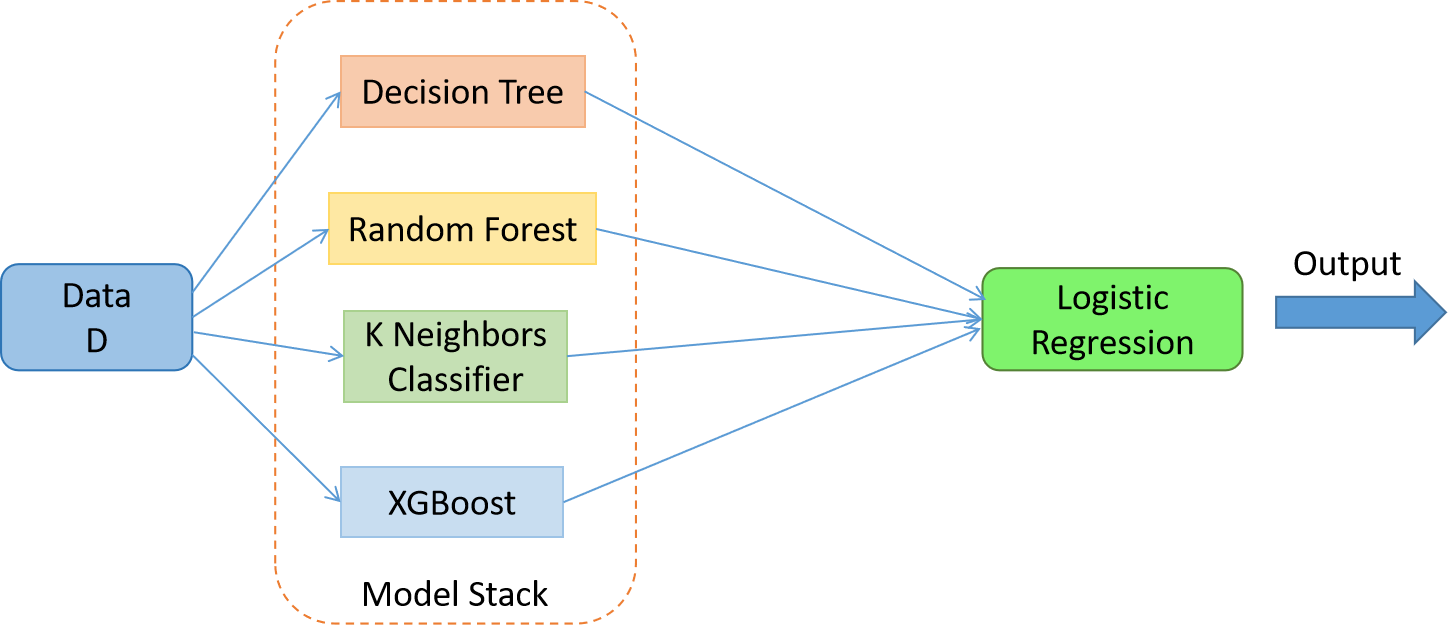




Better known as **Stacking Generalization**, it is a method introduced by [David H. Wolpert in 1992](https://www.sciencedirect.com/science/article/abs/pii/S0893608005800231) [3] where the key is to reduce the generalization error of different generalizers (i.e. ML models). The general idea of the **Stacking Generalization** method is the generation of a **Meta-Model**. Such a **Meta-Model** is made up of the predictions of a set of *ML base models* (i.e. weak learners) through the k-fold cross validation technique. Finally, the **Meta-Model** is trained with an additional ML model (which is commonly known as the "*final estimator*" or "*final learner*").

The **Stacking Generalization** method is commonly composed of 2 training stages, better known as "*level 0*" and "*level 1*". It is important to mention that it can be added as many levels as necessary. However, in practice it is common to use only 2 levels. The aim of the first stage (*level 0*) is to generate the training data for the *meta-model*, this is carried out by implementing k-fold cross validation for each "*weak learner*" defined in the first stage. The predictions of each one of these"*weak learners*" are "*stacked*" in order to build such such "*new training set*" (the *meta-model*). The aim of the second stage (*level 1*) is to train the meta-model, such training is carried out through an already determined "*final learner*".





Stacking (K-Fold method)

**ensemble learning** is one of the most powerful techniques used to improve the accuracy, robustness, and generalization of models. Rather than relying on a single predictive model, ensemble learning combines the predictions of multiple models to create a more accurate and reliable final prediction. The intuition is that multiple models, or **weak learners**, can correct each other’s errors, resulting in a more robust **strong learner**.

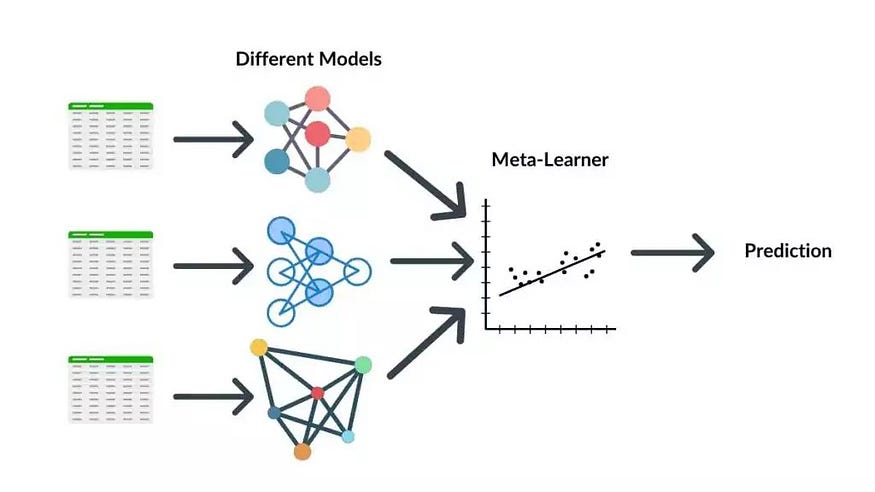
Some advantages of ensemble learning include:

* **Improved accuracy**: By averaging or combining the predictions from multiple models, ensembles often outperform individual models.
* **Reduced overfitting**: Ensemble methods help reduce overfitting by smoothing out noisy predictions.
* **Model diversity**: Ensembles make use of multiple algorithms or variations of the same algorithm, which can capture different aspects of the data.

To learn more about [**bagging and boosting, follow this blog**](https://medium.com/@abhishekjainindore24/a-comprehensive-guide-to-ensemble-techniques-bagging-and-boosting-fa276e28da9f)

**3. Stacking**

Press enter or click to view image in full size



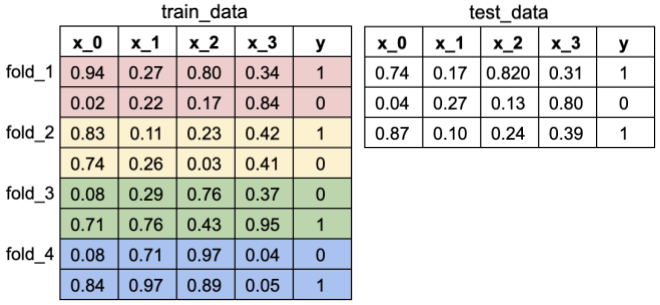
Stacking technique workflow

**Stacking** is a more sophisticated ensemble technique that involves combining different types of models (often called base learners) to improve performance. The idea behind stacking is to leverage the strengths of several models by training a **meta-model** (often called a second-level model) that learns to make predictions based on the outputs of the base models.

**How Stacking Works:**

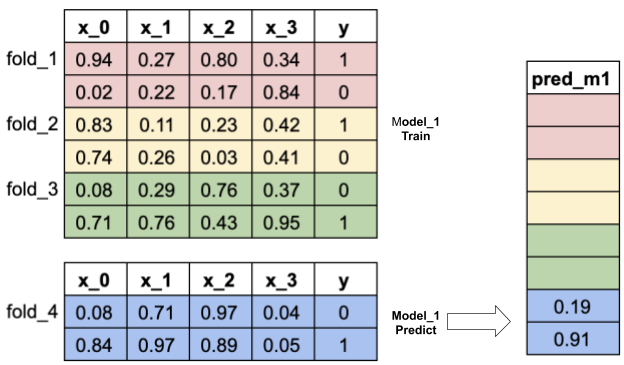
1. Train multiple base models (e.g., decision trees, logistic regression, SVMs) on the training data.
2. The predictions from these base models are fed into a **meta-model** (typically a more complex model like a neural network or linear regression).
3. The meta-model learns to combine the predictions of the base models and outputs the final prediction.

**Step 1**: You have Train Data and Test Data. Assume we are using 4-fold cross validation to train base models, the train\_data is then divided into 4 parts.

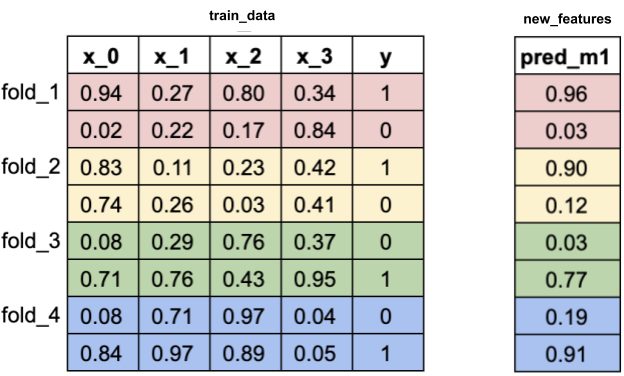


Training data (4-fold) and Testing data

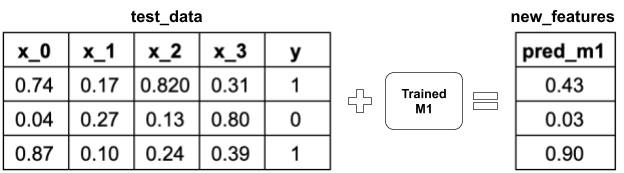
**Step 2**: Using the 4-part train\_data, the 1st base model (assuming its a decision tree) is fitted on 3 parts and predictions are made for the 4th part. This is done for each part of the training data. At the end, all instance from training data will have a prediction. This creates a new feature for train\_data, call it pred\_m1 (predictions model 1).



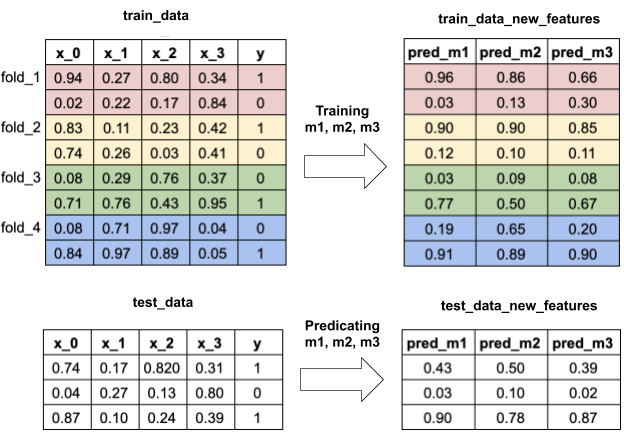
Model 1 training and prediction using 4-fold cross validation



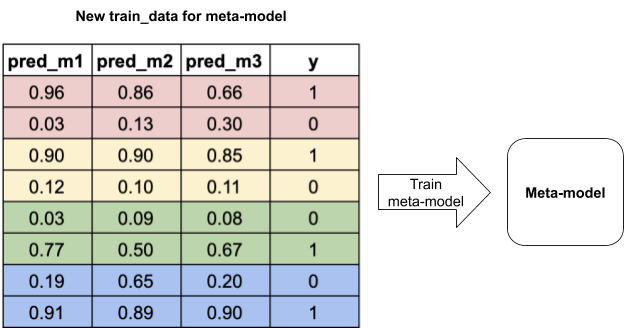
**Step 3**: Model 1 (decision tree) is then fitted on the whole training data — no folding is needed this time. The trained model will be used to predict Test Data. So test\_data will also have pred\_m1.



**Step 4**: Step 2 to 3 are repeated for the 2nd model (e.g KNN) and the 3rd model (e.g. SVM). These will give both train\_data and test\_data two more features from the predictions, pred\_m2 and pred\_m3.



**Step 5**: Now, to train the meta model (assume it’s a logistic regression), we use only the newly added features from the base models, which are [pred\_m1, pred\_m2, pred\_m3]. Fit this meta model on train\_data.



**Step 6**: The final prediction for test\_data is given by the trained meta model.

**Example:**

In a classification problem, you might train three models: a decision tree, an SVM, and a k-nearest neighbors model. The outputs of these models are then used as features for a meta-model (e.g., a logistic regression), which makes the final classification decision.

**Advantages of Stacking:**

* Combines models with different strengths to improve overall performance.
* Often leads to better performance than using any single model.

**Stacking**

Example : Lets say we have 4 input features (x1,x2,x3,x4) and 1 output feature (y), And we have 8 records in the dataset

For stacking, lets say the model m1 is trained. Let the k fold size be 2. Which means that for first iteration it will take first 6 records for training and then give the output for the last 2 records. In the second iteration it will take first 4 records, and last 2 records for training and give output for 5th and 6th record. Similarly 2 more iterations will happen. And the prediction by the model on each of the k fold dataset be p1,p2,p3,p4,p5,p6,p7 and p8.

Now similarly other lets say 2 models will be trained in a similar way and let its prediction be p9,p10,p11,p12,p13,p14,p15,p16 and p17,p18,p19,p20,p21,p22,p23,p24.

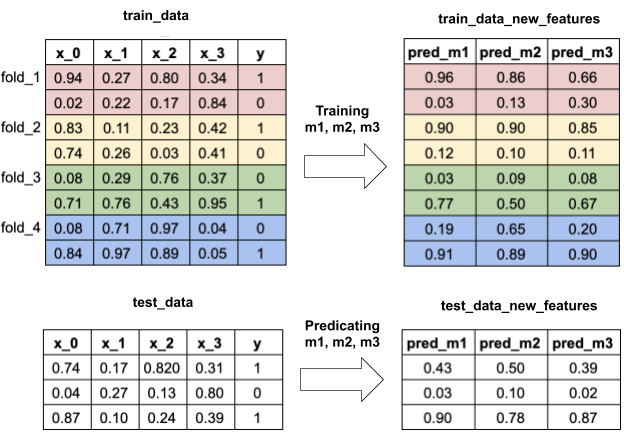
Here note that p1,p9 and p17 are predictions made by different model on the same record and similarly for p2,p10,p18 and so on and so forth

Now this data will be sent to the meta model to learn the relationship between the models prediction and the actual y value.

One such row for the dataset which will be sent to the meta model will look like (p1,p9,p17 as input feature and the output as the actual output value of the record on which the each of the 3 model has given its prediction)

**Key Characteristics of Stacking:**

* **Training Data**: The original dataset is divided into **k-folds** for cross-validation:
* Each base model is trained on k−1 folds and makes predictions on the remaining fold. This process is repeated k times, with each fold being used as a validation set once.
* Predictions from each base model across all folds are collected and used to train the meta-model.
* **Meta-Model**: Trained on predictions from the base models across all folds. This meta-model learns to combine the base models’ predictions to make the final prediction.



Code

|  |
| --- |
| def StackingClassifier(self): |
|  |
| # Define weak learners |
| weak\_learners = [('dt', DecisionTreeClassifier()), |
| ('knn', KNeighborsClassifier()), |
| ('rf', RandomForestClassifier()), |
| ('gb', GradientBoostingClassifier()), |
| ('gn', GaussianNB())] |
|  |
| # Finaler learner or meta model |
| final\_learner = LogisticRegression() |
|  |
| train\_meta\_model = None |
| test\_meta\_model = None |
|  |
| # Start stacking |
| for clf\_id, clf in weak\_learners: |
| # Predictions for each classifier based on k-fold |
| predictions\_clf = self.k\_fold\_cross\_validation(clf) |
|  |
| # Predictions for test set for each classifier based on train of level 0 |
| test\_predictions\_clf = self.train\_level\_0(clf) |
|  |
| # Stack predictions which will form |
| # the inputa data for the data model |
| if isinstance(train\_meta\_model, np.ndarray): |
| train\_meta\_model = np.vstack((train\_meta\_model, predictions\_clf)) |
| else: |
| train\_meta\_model = predictions\_clf |
|  |
| # Stack predictions from test set |
| # which will form test data for meta model |
| if isinstance(test\_meta\_model, np.ndarray): |
| test\_meta\_model = np.vstack((test\_meta\_model, test\_predictions\_clf)) |
| else: |
| test\_meta\_model = test\_predictions\_clf |
|  |
| # Transpose train\_meta\_model |
| train\_meta\_model = train\_meta\_model.T |
|  |
| # Transpose test\_meta\_model |
| test\_meta\_model = test\_meta\_model.T |
|  |
| # Training level 1 |
| self.train\_level\_1(final\_learner, train\_meta\_model, test\_meta\_model) |
|  |

from numpy import mean from numpy import std from sklearn.datasets import make\_classification from sklearn.model\_selection import cross\_val\_score from sklearn.model\_selection import RepeatedStratifiedKFold 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 from sklearn.ensemble import StackingClassifier from matplotlib import pyplot from sklearn.datasets import load\_wine,load\_iris from matplotlib.pyplot import figure figure(num=2, figsize=(16, 12), dpi=80, facecolor='w', edgecolor='k') # get a stacking ensemble of models def get\_stacking(): # define the base models level0 = list() level0.append(('lr', LogisticRegression())) level0.append(('knn', KNeighborsClassifier())) level0.append(('cart', DecisionTreeClassifier())) level0.append(('svm', SVC())) level0.append(('bayes', GaussianNB())) # define meta learner model level1 = LogisticRegression() # define the stacking ensemble model = StackingClassifier(estimators=level0, final\_estimator=level1, cv=5) return model # get a list of models to evaluate def get\_models(): models = dict() models['LogisticRegression'] = LogisticRegression() models['KNeighborsClassifier'] = KNeighborsClassifier() models['Decision tree'] = DecisionTreeClassifier() models['svm'] = SVC() models['GaussianNB'] = GaussianNB() models['stacking'] = get\_stacking() return models # evaluate a give model using cross-validation def evaluate\_model(model): cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1) scores = cross\_val\_score(model, X, y, scoring='accuracy', cv=cv, n\_jobs=-1, error\_score='raise') scores1 = cross\_val\_score(model, X1, y1, scoring='accuracy', cv=cv, n\_jobs=-1, error\_score='raise') return scores,scores1 # define dataset X,y = load\_wine().data,load\_wine().target X1,y1= load\_iris().data,load\_iris().target # get the models to evaluate models = get\_models() # evaluate the models and store results results, names, results1 = list(), list(),list() for name, model in models.items(): scores,scores1= evaluate\_model(model) results.append(scores) results1.append(scores1) names.append(name) print('>%s -> %.3f (%.3f)---Wine dataset' % (name, mean(scores), std(scores))) print('>%s -> %.3f (%.3f)---Iris dataset' % (name, mean(scores1), std(scores1))) # plot model performance for comparison pyplot.rcParams["figure.figsize"] = (15,6) pyplot.boxplot(results, labels=[s+"-wine" for s in names], showmeans=True) pyplot.show() pyplot.boxplot(results1, labels=[s+"-iris" for s in names], showmeans=True) pyplot.show()