Laboratory Session 6

Bagging –Boosting-Stacking

Materials:

-Tutorial iris dataset links:

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html

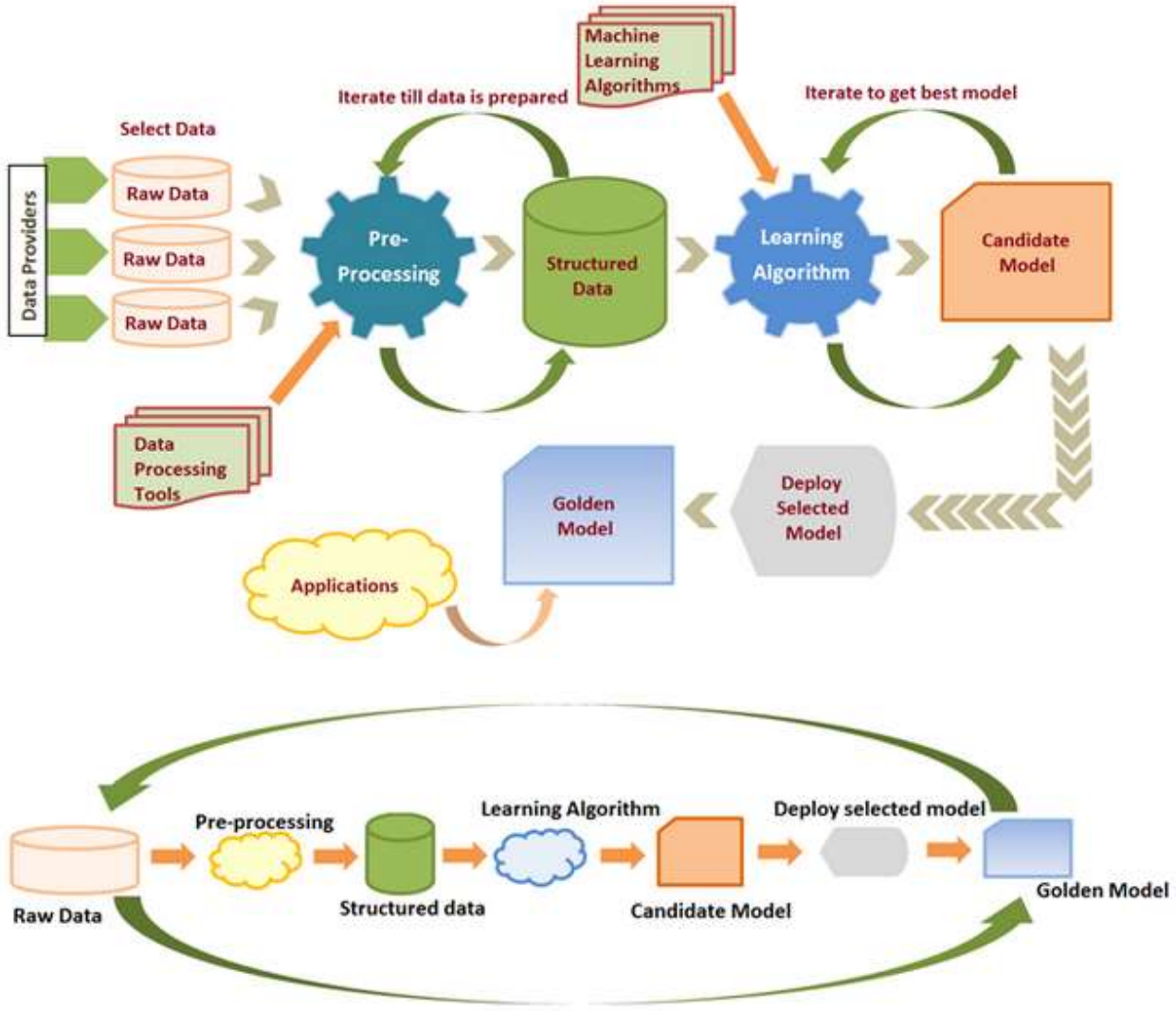
-Tutorial Ensemble learning: [https://www.datacamp.com/community/tutorials/ensemble-learning-python]

-Videos: 1. Scikit Learn Ensemble Learning, Bootstrap Aggregating (Bagging) and Boosting: https://www.youtube.com/watch?v=X3Wbfb4M33w 2. Pruning trees and bagging: https://www.youtube.com/watch?v=vZMnAyqYzsY 3. Bootstrapping, Bagging and Random Forests: https://www.youtube.com/watch?v=3R0AW-vrPEw

Stacking: https://blogs.sas.com/content/subconsciousmusings/2017/05/18/stacked-ensemble-models-win-data-science-competitions/

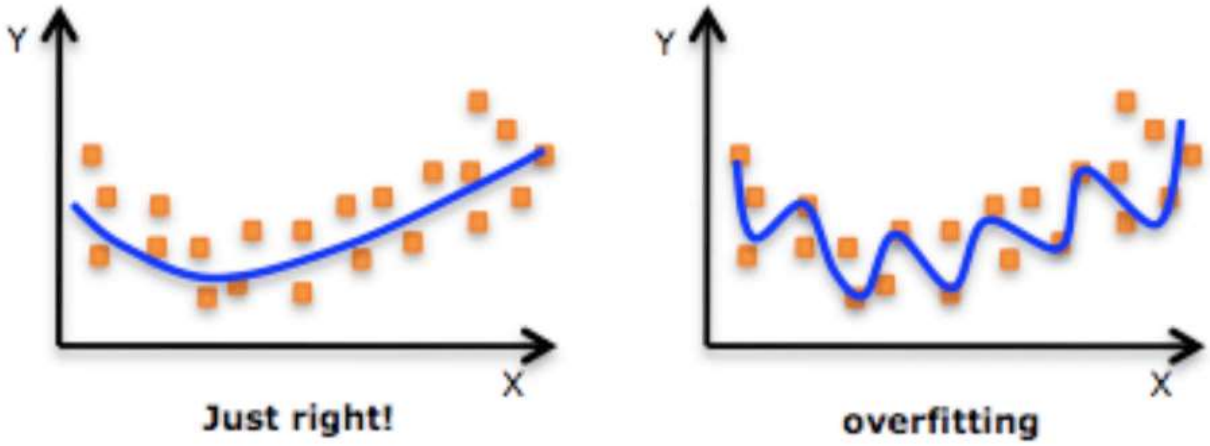
1. **Machine Learning Scenario**

- Candidate model means the first most appropriate model that we get, but still needs to be massaged
- But do we get only one candidate model? Of course not, since this is an iterative process, we do not actually know what the best candidate model is until we again and again produce several candidate models through the iterative process. We do it until we get the model that is good enough to be deployed. Once the model is deployed, applications start making use of it, so there is iteration at small levels and at the largest level as well.
- We need to repeat the entire process again and again and re-create the model at regular intervals. The reason again for this process is very simple, it's because the scenarios and factors change and we need to have our model up to date and real all the time. This could eventually also mean to process new data or applying new algorithms altogether.

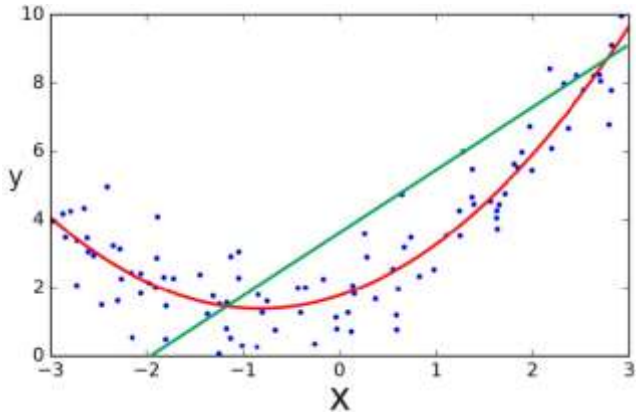


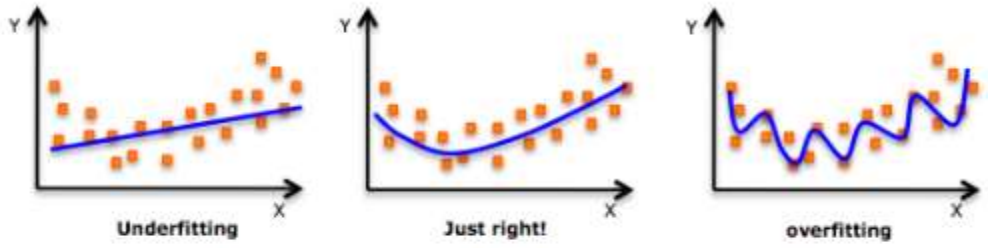
2. Overfitting/Underfitting

- **Overfitting** means that model we trained has trained "too well and is now, well", fit too closely to the training dataset
- This usually happens when the model is too complex (i.e. too many features/variables compared to the number of observations). This model will be very accurate on the training data but will probably be very not accurate on untrained or new data. It is because this model is not generalized (or not AS generalized)



- In contrast to **overfitting**, when a model is **underfitted**, it means that the model does not fit the training data and therefore misses the trends in the data
- It also means the model cannot be generalized to new data. As you probably guessed (or figured out!), this is usually the result of a very simple model (not enough predictors/independent variables). It could also happen when, for example, we fit a linear model (like linear regression) to data that is not linear. It almost goes without saying that this model will have poor predictive ability (on training data and can't be generalized to other data)

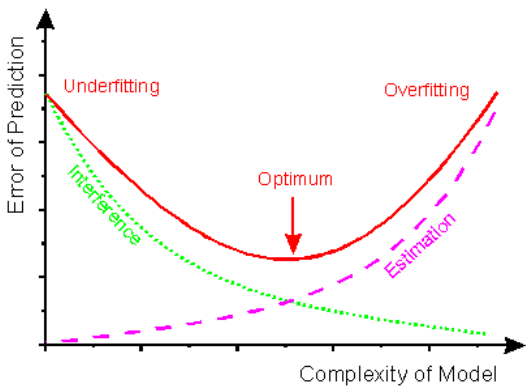


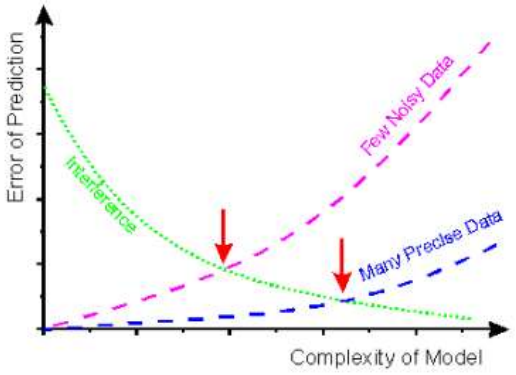


An example of overfitting, underfitting and a model that's "just right!"

- The error of prediction is composed of two main contributions, the remaining **interference error and the estimation error**
- The **interference error** is the systematic error (bias) due to unmodeled interference in the data, as the calibration model is not complex enough to capture all the interferences of the relationship between sensor responses and analytes. The **estimation error** is caused by modeling measured random noise of various kinds.

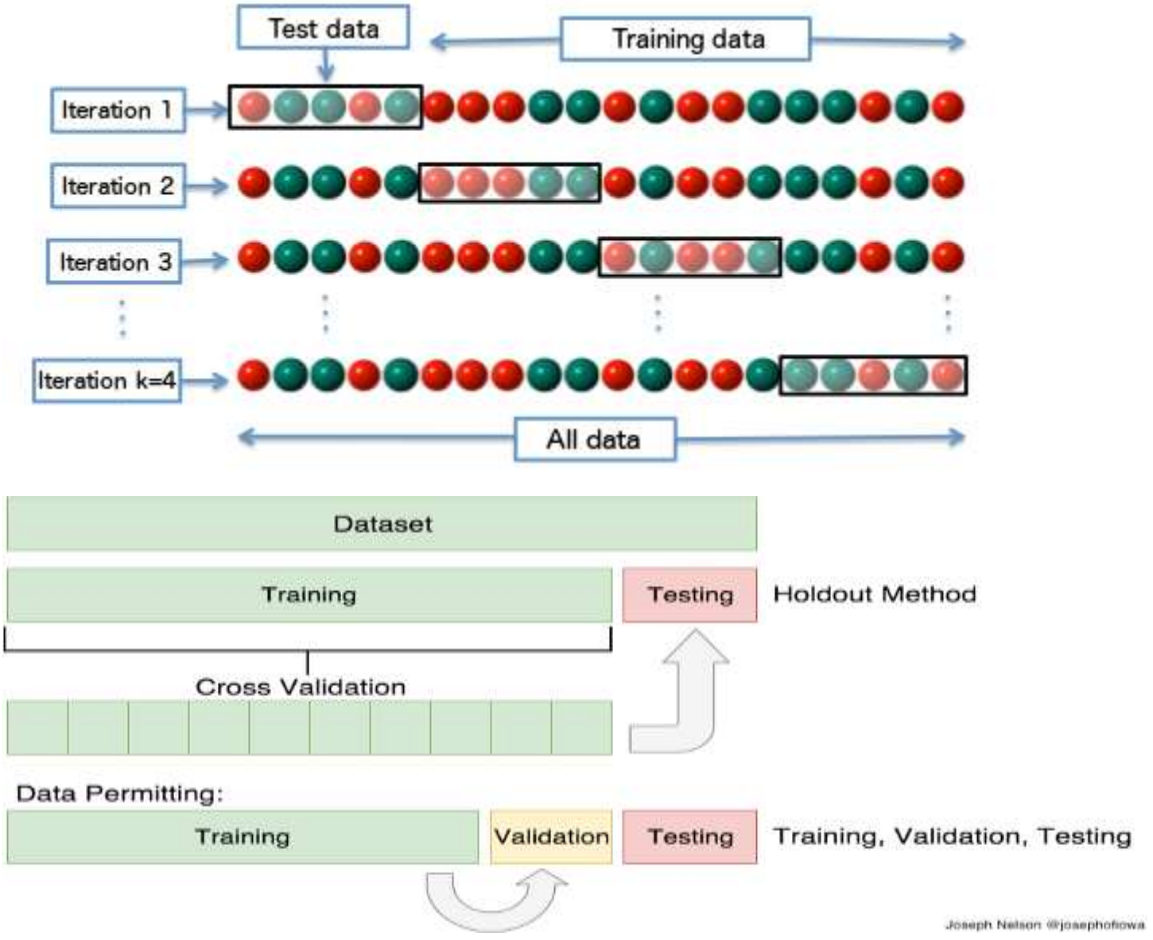
- Estimation −>−> find unknown values (estimates) for subject of interest
- Statistical Inference −>−> use the probability distribution of subject of interest to make probabilistic conclusions



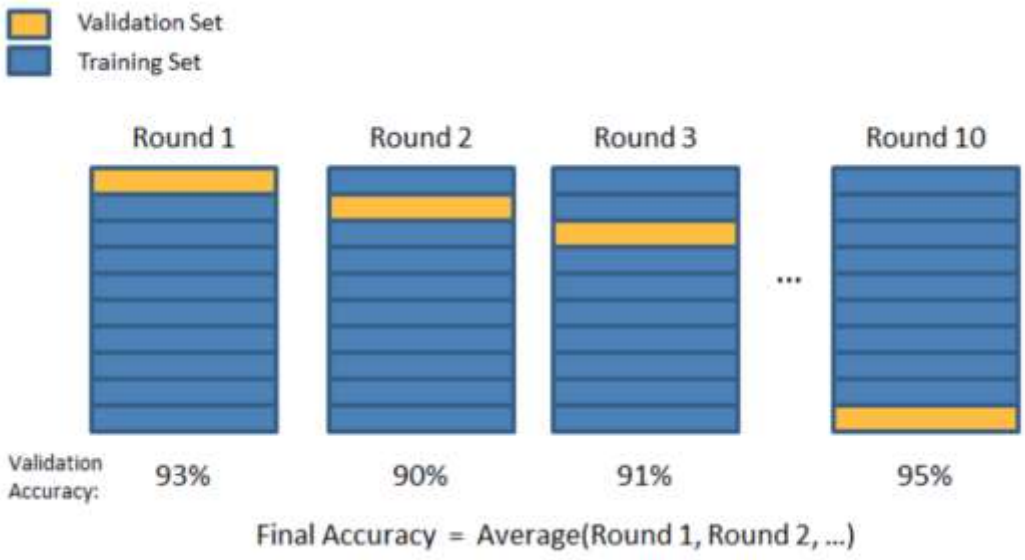


3. Cross Validation

- **Cross-validation**, sometimes called **rotation estimation**, or **out-of-sample testing** is any of various similar model validation techniques for assessing how the results of a statistical analysis will generalize to an independent data set
- It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. In a prediction problem, a model is usually given a dataset of known data on which training is run (**training dataset**), and a dataset of unknown data (or first seen data) against which the model is tested (called the **validation dataset or testing set**)
- We split our data into k subsets, and train on k-1 one of those subset. What we do is to hold the last subset for test.



- There are a bunch of cross validation methods, I'll go over two of them: the first is **K-Folds Cross Validation** and the second is **Leave One Out Cross Validation (LOOCV)**
- In **K-Folds Cross Validation** we split our data into k different subsets (or folds). We use k-1 subsets to train our data and leave the last subset (or the last fold) as test data. We then average the model against each of the folds and then finalize our model. After that we test it against the test set.



- **Leave One Out Cross Validation (LOOCV)**: the number of folds (subsets) equals to the number of observations we have in the dataset. We then average ALL of these folds and build our model with the average. We then test the model against the last fold. Because we would get a big number of training sets (equals to the number of samples), this method is very computationally expensive and should be used on small datasets. If the dataset is big, it would most likely be better to use a different method, like kfold

4. Ensemble learning

4.1 What is Ensemble learning?

In the world of Statistics and Machine Learning, Ensemble learning techniques attempt to make the performance of the predictive models better by improving their accuracy. Ensemble Learning is a process using which multiple machine learning models (such as classifiers) are strategically constructed to solve a particular problem.

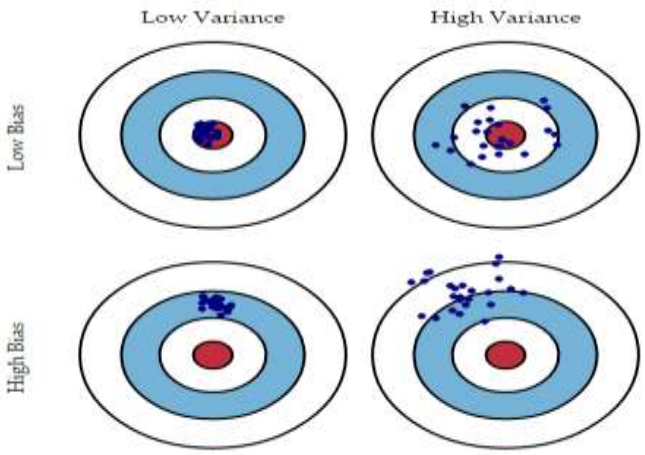
4.2 Model error and reducing this error with Ensembles:

The error emerging from any machine model can be broken down into three components mathematically. Following are these component:

Bias + Variance + Irreducible error

Bias error is useful to quantify how much on an average are the predicted values different from the actual value. A high bias error means we have an under-performing model which keeps on missing essential trends.

Variance on the other side quantifies how are the prediction made on the same observation different from each other. A high variance model will over-fit on your training population and perform poorly on any observation beyond training. Following diagram will give you more clarity (Assume that red spot is the real value, and blue dots are predictions):



Typically, as you increase the complexity of your model, you will see a reduction in error due to lower bias in the model. However, this only happens until a particular point. As you continue to make your model more complex, you end up over-fitting your model, and hence your model will start suffering from the high variance.

Now that you are familiar with the basics of Ensemble learning let's look at different Ensemble learning techniques: Although there are several types of Ensemble learning methods, the following three are the most-used ones in the industry: Bagging, Boosting, and Stacking.

4.3 Voting

- Ensemble methods are techniques that create multiple models and then combine them to produce improved results. Ensemble methods usually produces more accurate solutions than a single model would.
- **Majority Voting**: Every model makes a prediction (votes) for each test instance and the final output prediction is the one that receives more than half of the votes. If none of the predictions get more than half of the votes, we may say that the ensemble method could not make a stable prediction for this instance. This method is called “plurality voting”.
- **Weighted Voting**: Unlike majority voting, where each model has the same rights, we can increase the importance of one or more models. In weighted voting you

count the prediction of the better models multiple times. Finding a reasonable set of weights is up to you.

- **Simple Averaging**: In simple averaging method, for every instance of test dataset, the average predictions are calculated. This method often reduces overfit and creates a smoother regression model
- **Weighted Averaging**: Weighted averaging is a slightly modified version of simple averaging, where the prediction of each model is multiplied by the weight and then their average is calculated

Hard voting

Predictions:

> Classifier 1 predicts class A
>
> Classifier 2 predicts class B
>
> Classifier 3 predicts class B

2/3 classifiers predict class B, so **class B is the ensemble decision**.

Soft voting

Predictions (identical to the earlier example, but now in terms of probabilities. Shown only for class A here because the problem is binary):

> Classifier 1 predicts class A with probability 99%
>
> Classifier 2 predicts class A with probability 49%
>
> Classifier 3 predicts class A with probability 49%

The average probability of belonging to class A across the classifiers is `(99 + 49 + 49) / 3 = 65.67%`. Therefore, **class A is the ensemble decision**.

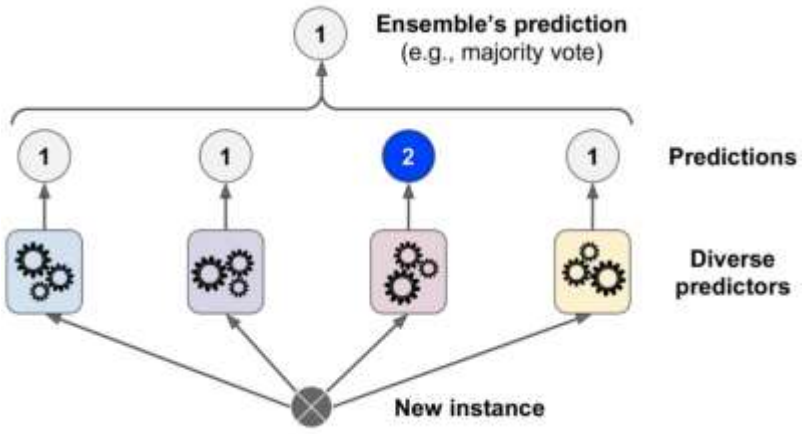
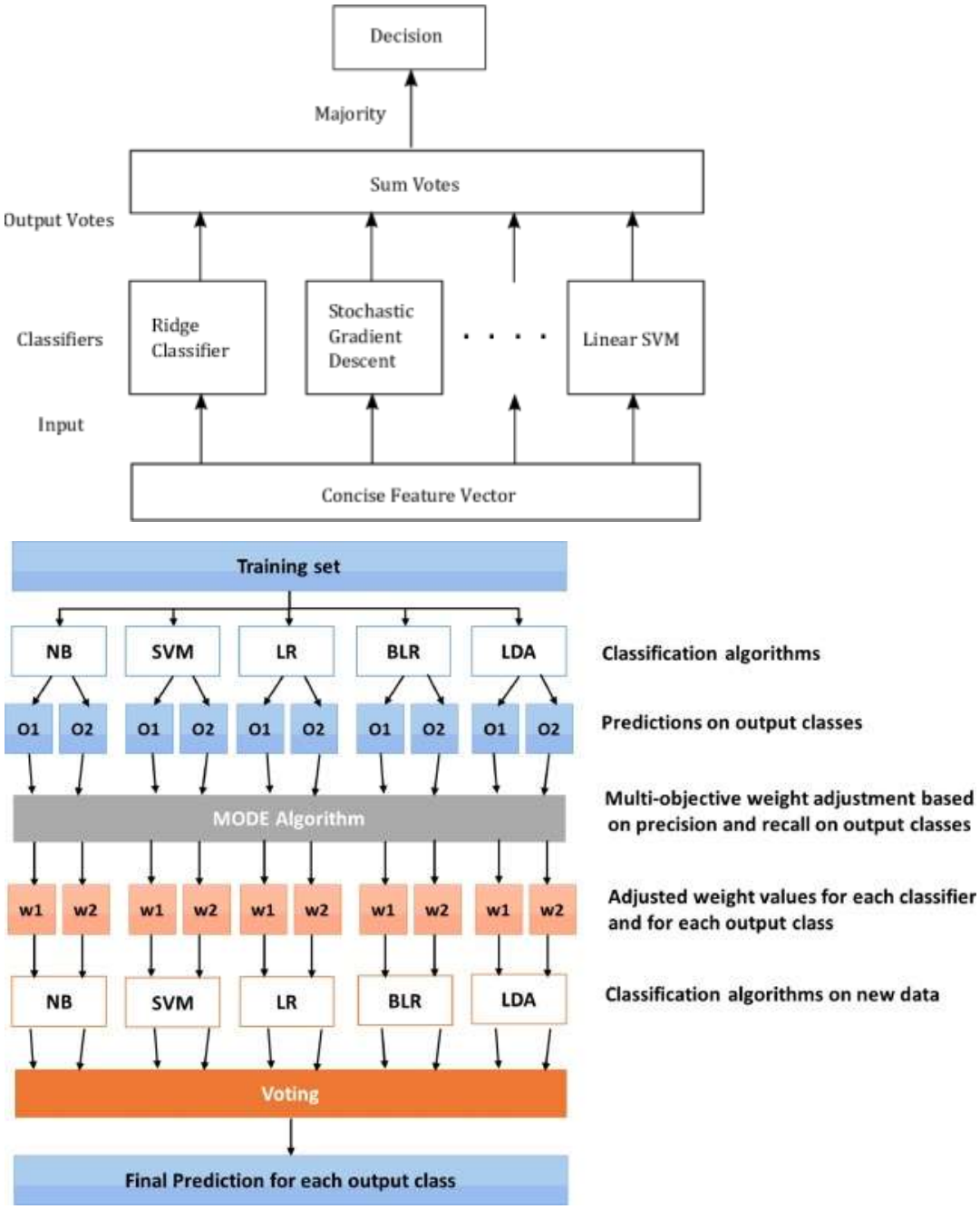


Figure 7-2. Hard voting classifier predictions

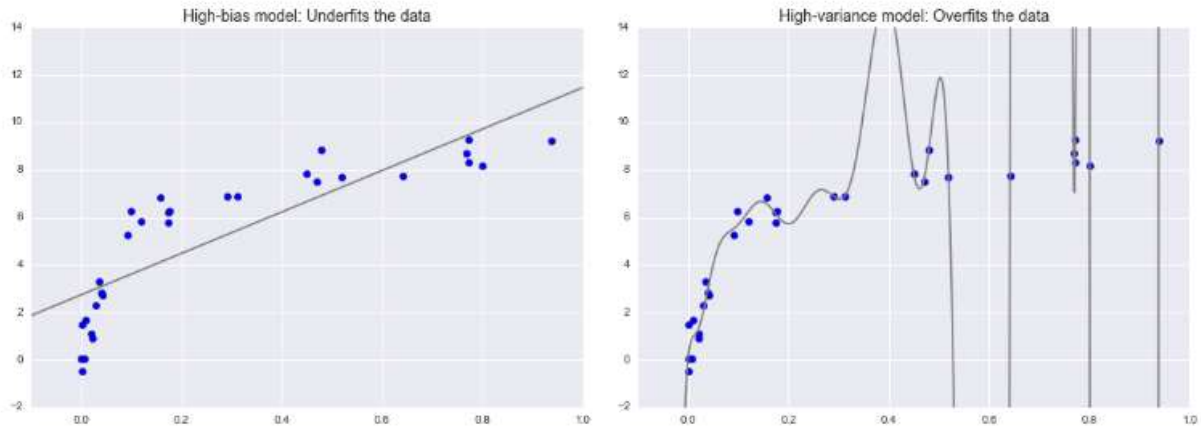


4.4. Validation Curve

- Fundamentally, the question of "the best model" is about finding a sweet spot in the tradeoff between *bias* and *variance*.
- The model on the left attempts to find a straight-line fit through the data. Because the data are intrinsically more complicated than a straight line, the straight-line model will never be able to describe this dataset well. Such a model is said

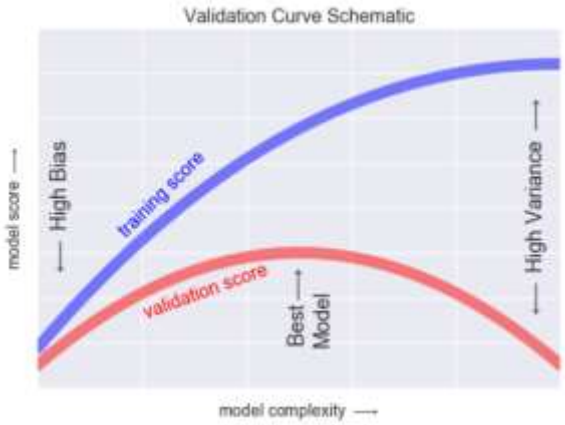
to *underfit* the data: that is, it does not have enough model flexibility to suitably account for all the features in the data; another way of saying this is that the model has high *bias*.

- The model on the right attempts to fit a high-order polynomial through the data. Here the model fit has enough flexibility to nearly perfectly account for the fine features in the data, but even though it very accurately describes the training data, its precise form seems to be more reflective of the particular noise properties of the data rather than the intrinsic properties of whatever process generated that data. Such a model is said to *overfit* the data: that is, it has so much model flexibility that the model ends up accounting for random errors as well as the underlying data distribution; another way of saying this is that the model has high *variance*.



-For high-bias models, the performance of the model on the validation set is similar to the performance on the training set.

-For high-variance models, the performance of the model on the validation set is far worse than the performance on the training set.



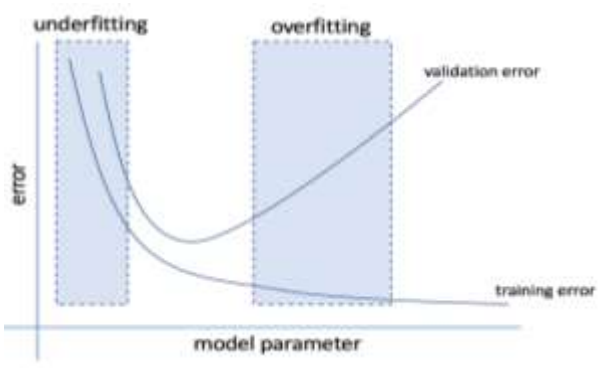
The diagram show here is often called a validation curve, and we see the following essential features:

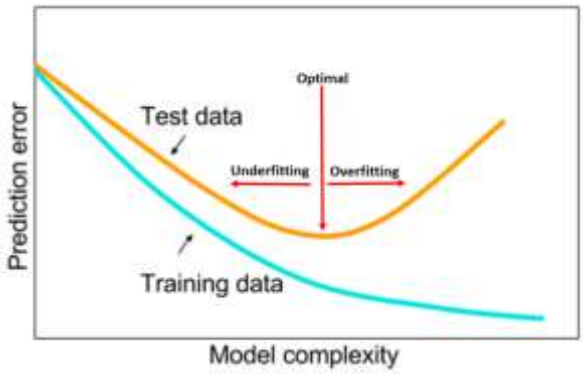
-The training score is everywhere higher than the validation score. This is generally the case: the model will be a better fit to data it has seen than to data it has not seen.

-For very low model complexity (a high-bias model), the training data is under-fit, which means that the model is a poor predictor both for the training data and for any previously unseen data.

-For very high model complexity (a high-variance model), the training data is over-fit, which means that the model predicts the training data very well, but fails for any previously unseen data.

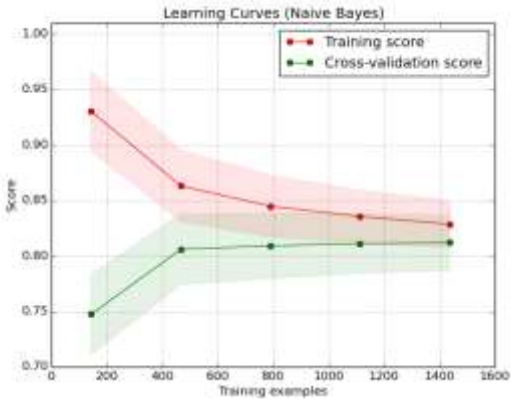
-For some intermediate value, the validation curve has a maximum. This level of complexity indicates a suitable trade-off between bias and variance.

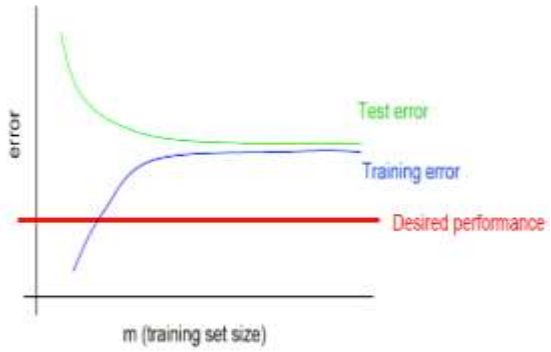




4.5 Learning Curve

- The learning curve represents the evolution of the training and test errors as you increase the size of your training set.
    - The training error increases as you increase the size of your dataset, because it becomes harder to fit a model that accounts for the increasing complexity/variability of your training set.
    - The test error decreases as you increase the size of your dataset, because the model is able to generalize better from a higher amount of information.
- As you can see on the rightmost part of the plot, the two lines in the plot tend to reach and asymptote. Therefore, you eventually will reach a point in which increasing the size of your dataset will not have an impact on your trained model.
- The distance between the test error and training error asymptotes is a representation of your model's overfitting. But more importantly, this plot is saying whether you need more data. Basically, if you represent test and training error for increasing larger subsets of your training data, and the lines do not seem to be reaching an asymptote, you should keep collecting more data.

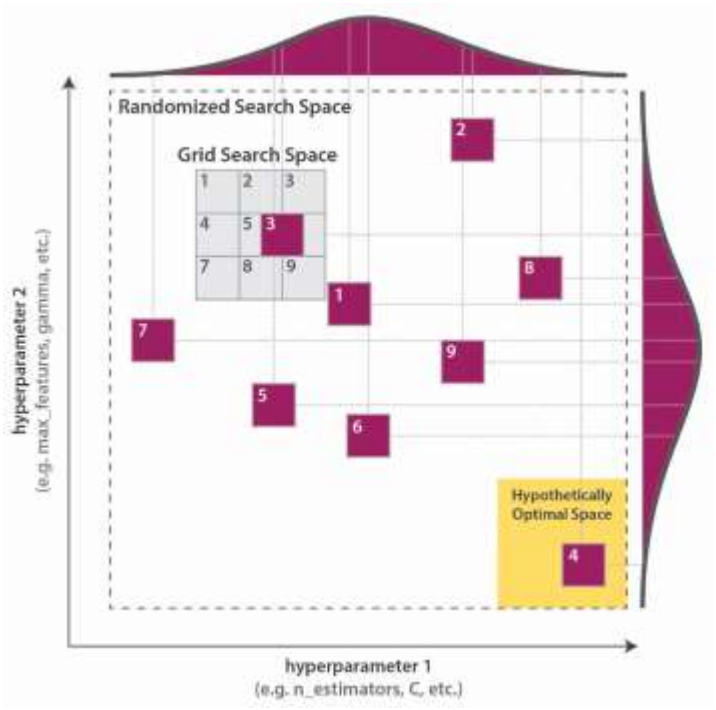


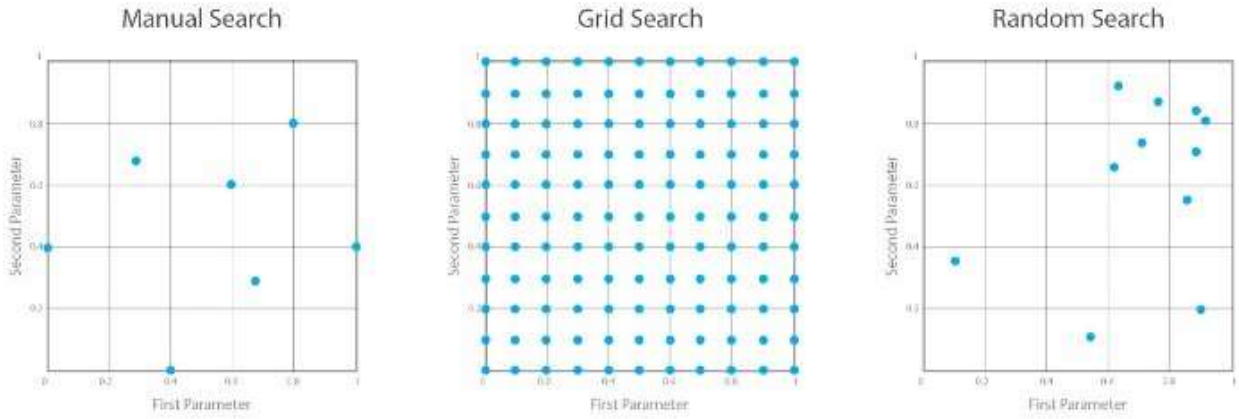


4.6 Grid Search

- A machine learning model has two types of parameters. The first type of parameters are the parameters that are learned through a machine learning model while the second type of parameters are the hyper parameter that we pass to the machine learning model

- Therefore, instead of randomly selecting the values of the parameters, a better approach would be to develop an algorithm which automatically finds the best parameters for a particular model. Grid Search is one such algorithm





```
grid_param = {
    'n_estimators': [100, 300, 500, 800, 1000],
    'criterion': ['gini', 'entropy'],
    'bootstrap': [True, False]
}
```

Take a careful look at the above code. Here we create `grid_param` dictionary with three parameters `n_estimators`, `criterion`, and `bootstrap`. The parameter values that we want to try out are passed in the list. For instance, in the above script we want to find which value (out of 100, 300, 500, 800, and 1000) provides the highest accuracy.

```
gd_sr = GridSearchCV(estimator=classifier,
                     param_grid=grid_param,
                     scoring='accuracy',
                     cv=5,
                     n_jobs=-1)
```

Once the `GridSearchCV` class is initialized, the last step is to call the `fit` method of the class and pass it the training and test set, as shown in the following code:

```
gd_sr.fit(X_train, y_train)
```

This method can take some time to execute because we have 20 combinations of parameters and a 5-fold cross validation. Therefore the algorithm will execute a total of 100 times.

Once the method completes execution, the next step is to check the parameters that return the highest accuracy. To do so, print the `sr.best_params_` attribute of the `GridSearchCV` object, as shown below:

```
best_parameters = gd_sr.best_params_
print(best_parameters)
```

Output:

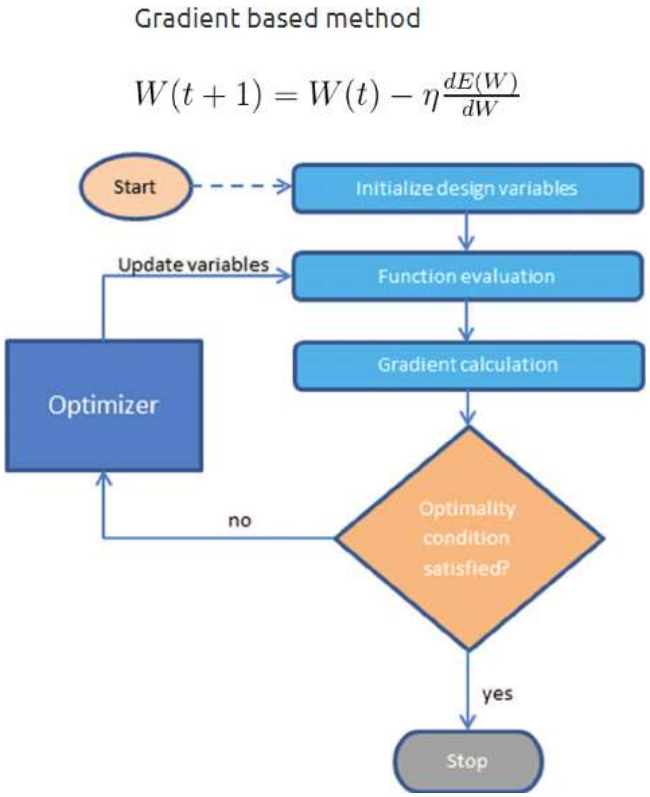
```
{'bootstrap': True, 'criterion': 'gini', 'n_estimators': 1000}
```

The result shows that the highest accuracy is achieved when the `n_estimators` are 1000, `bootstrap` is `True` and `criterion` is "gini".

4.7 Gradient Based Optimization

Link: http://rasbt.github.io/mlxtend/user_guide/classifier/LogisticRegression/

- Using the Gradient Decent optimization algorithm, the weights are updated incrementally after each epoch (= pass over the training dataset)..



- Compatible cost functions J(·)
  - Sum of squared errors (SSE)

$$J(\mathbf{w}) = \frac{1}{2}\sum_i (\text{target}^{(i)} - \text{output}^{(i)})^2$$

  - Logistic Cost (cross-entropy)

$$J(\mathbf{w}) = \sum_{i=1}^{m} -y^{(i)} log\left(\phi(z^{(i)})\right) - (1-y^{(i)}) log\left(1-\phi(z^{(i)})\right)$$

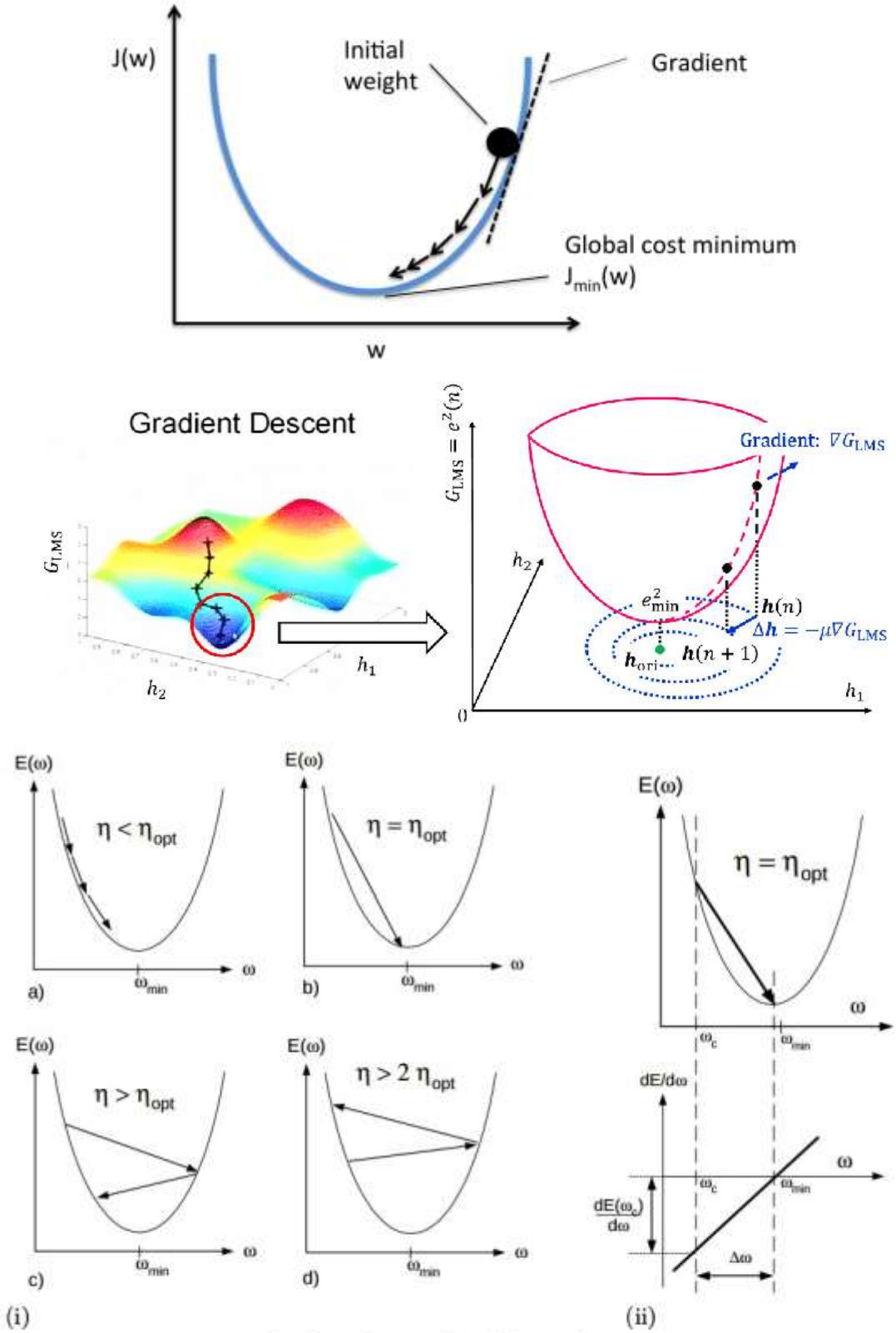


Fig. 6. Gradient descent for different learning rates.

- **Stochastic Gradient Descent (SGD)**

- **Link : http://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/**
    - In Gradient Descent optimization, we compute the cost gradient based on the complete training set; hence, we sometimes also call it *batch gradient descent*. In case of very large datasets, using Gradient Descent can be quite costly since we are only taking a single step for one pass over the training set -- thus, the larger the training set, the slower our algorithm updates the weights and the longer it may take until it converges to the global cost minimum
    - In Stochastic Gradient Descent (sometimes also referred to as *iterative* or *on-line* gradient descent), we **don't** accumulate the weight updates as we've seen above for Gradient Descent:
    - Here, the term "stochastic" comes from the fact that the gradient based on a single training sample is a "stochastic approximation" of the "true" cost gradient. Due to its stochastic nature, the path towards the global cost minimum is not "direct" as in Gradient Descent, but may go "zig-zag" if we are visuallizing the cost surface in a 2D space. However, it has been shown that Stochastic Gradient Descent almost surely converges to the global cost minimum if the cost function is convex (or pseudo-convex)

- for one or more epochs:
    - for each weight j
        - $w_j := w + \Delta w_j$, where: $\Delta w_j = \eta \sum_i (\text{target}^{(i)} - \text{output}^{(i)}) x_j^{(i)}$

Instead, we update the weights after each training sample:

- for one or more epochs, or until approx. cost minimum is reached:
    - for training sample i:
        - for each weight j
            - $w_j := w + \Delta w_j$, where: $\Delta w_j = \eta (\text{target}^{(i)} - \text{output}^{(i)}) x_j^{(i)}$

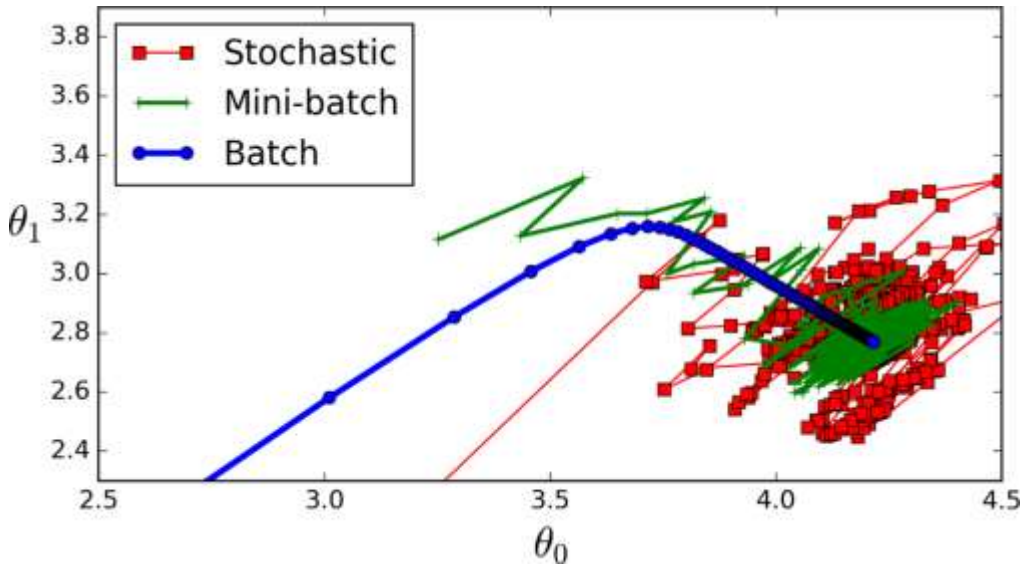
- **Mini-Batch Gradient Descent (MB-GD)**

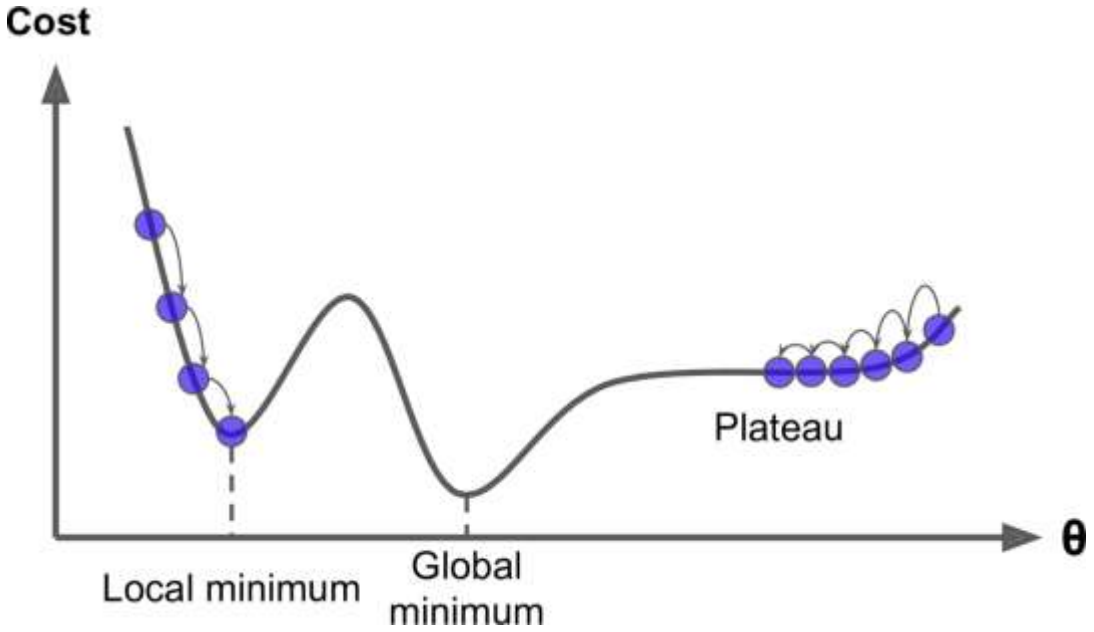
Mini-Batch Gradient Descent (MB-GD) a compromise between batch GD and SGD. In MB-GD, we update the model based on smaller groups of training samples; instead of computing the gradient from 1 sample (SGD) or all n training samples (GD), we compute the gradient from 1<k<n training samples (a common mini-batch size is k=50).

MB-GD converges in fewer iterations than GD because we update the weights more frequently; however, MB-GD let's us utilize vectorized operation, which typically results in a computational performance gain over SGD.

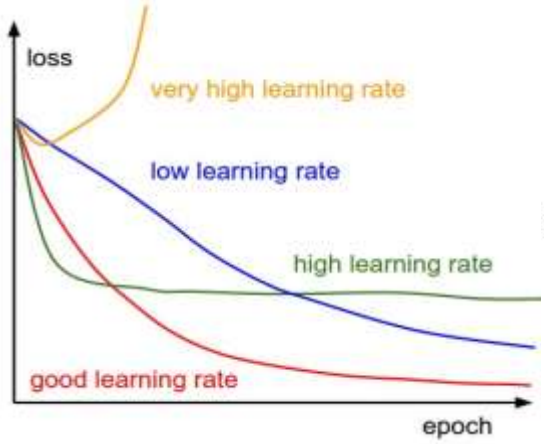
Learning Rates

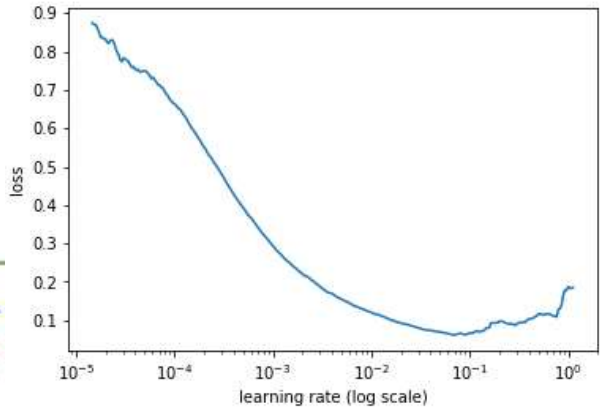
- An adaptive learning rate η: Choosing a decrease constant d that shrinks the learning rate over time: $\eta(t+1) := \eta(t)/(1+t\times d)$
- Momentum learning by adding a factor of the previous gradient to the weight update for faster updates: $\Delta \mathbf{w}_{t+1} := \eta \nabla J(\mathbf{w}_{t+1}) + \alpha \Delta w_t$





- Learning rate is a hyper-parameter that controls how much we are adjusting the weights of our network with respect the loss gradient. The lower the value, the slower we travel along the downward slope. While this might be a good idea (using a low learning rate) in terms of making sure that we do not miss any local minima, it could also mean that we'll be taking a long time to converge—especially if we get stuck on a plateau region.
- If we record the learning at each iteration and plot the learning rate (log) against loss; we will see that as the learning rate increase, there will be a point where the loss stops decreasing and starts to increase. In practice, our learning rate should ideally be somewhere to the left to the lowest point of the graph (as demonstrated in below graph). In this case, 0.001 to 0.01.





4.8 Dropout

- The primary idea is to randomly drop components of neural network (outputs) from a layer of neural network. This results in a scenario where at each layer more neurons are forced to learn the multiple characteristics of the neural network.
- Consider the case when there are 2 hidden layers with neurons A and B in one, and C and D in second. In this case, we have a scenario where during training, we want to train AC, AD, BC and BD all to learn the relation between input and output. Therefore, we have 4 models learning the same relation
- Therefore, we have 4 models learning the same relation. This number increases exponentially if we have more layers and more neurons. If we had 4 neurons and 2 layers, the number of possible models is 4C2 X 4C2 = 36, therefore we are taking average over 36 different models.
- For a 2 layer model with 100 neurons in each layer, this results in a scenario where we are taking average over billion possible models. As a result, the tendency to overfit is significantly reduced.

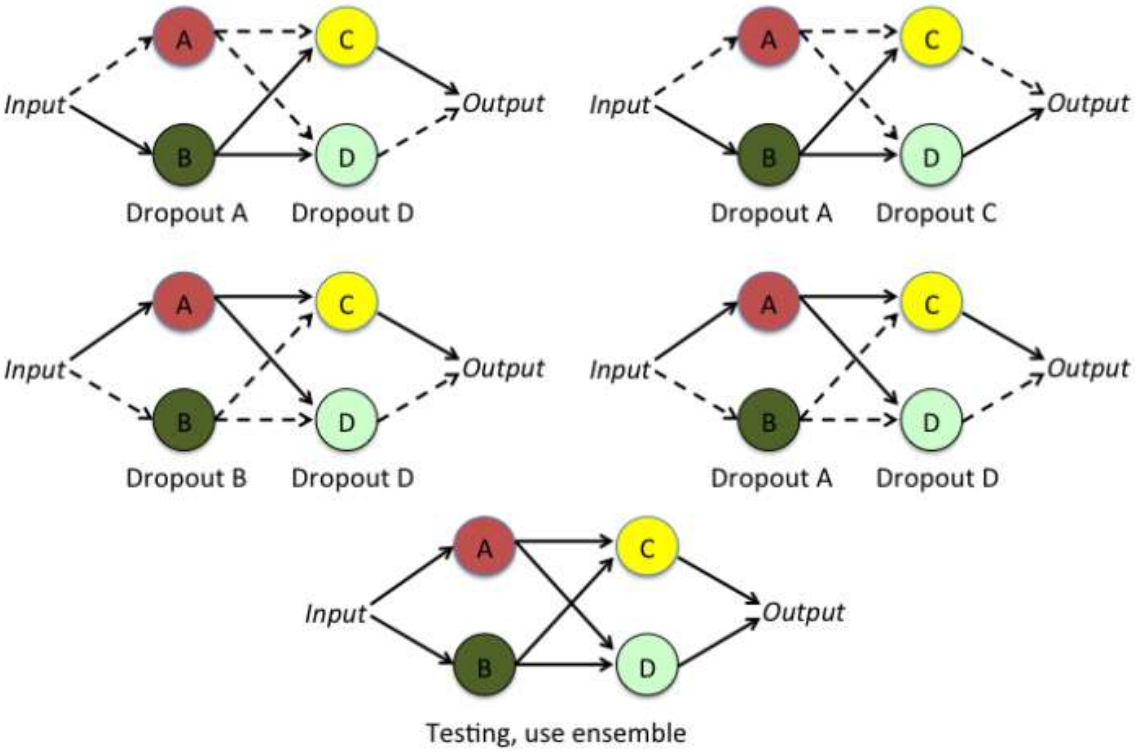


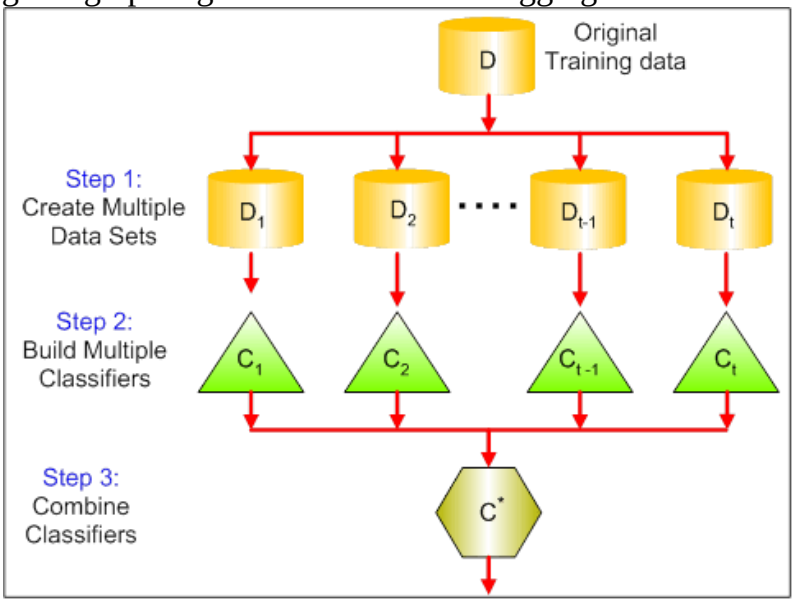
Illustration of dropout when there are 2 hidden layers and 2 hidden neurons.

4.9 Bagging based Ensemble learning (Bootstrap Aggregating)

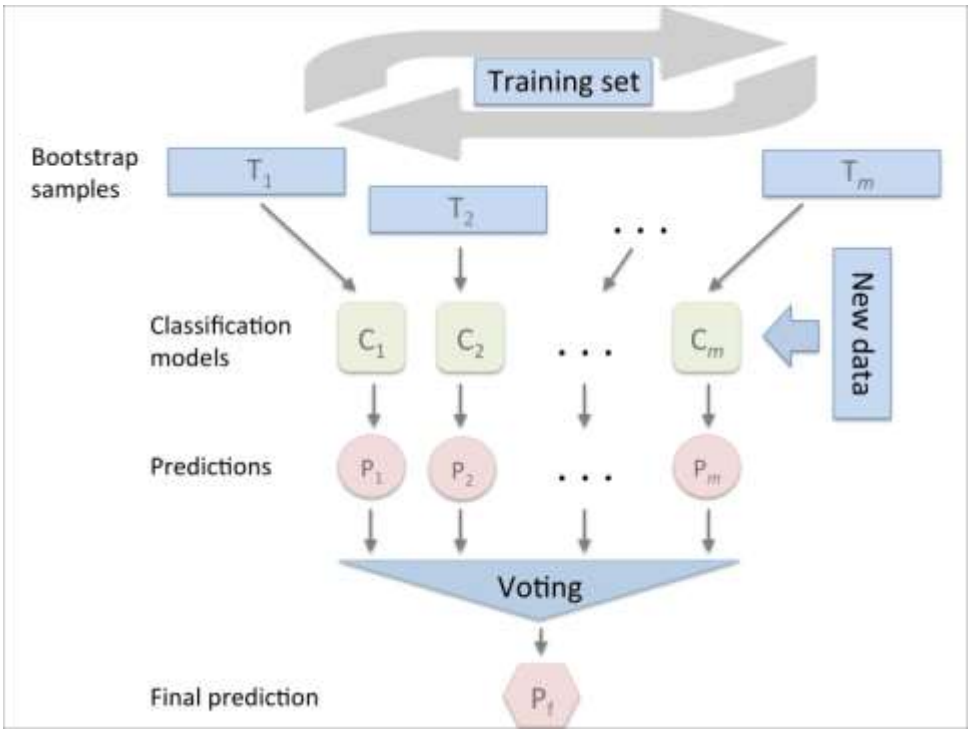
Bagging, which means bootstrap aggregation, is one of the simplest but most successful ensemble methods for improving unstable classification problems. For example, weak classifiers, such as decision tree algorithms, can be unstable, especially when the position of a training point changes slightly and can lead to a very different tree. This method is usually applied to decision tree algorithms, but it also can be used with other classification algorithms such as Naïve Bayes, nearest neighbour, rule induction, etc. The bagging technique is very useful for large and high-dimensional data, such as intrusion data sets, where finding a good model or classifier that can work in one step is impossible because of the complexity and scale of the problem. Bagging was first introduced by Leo Breiman to reduce the variance of a predictor. It uses multiple versions of a training set which is generated by a random draw with the replacement of N examples where N is the size of original training set. Each of these data sets is used to train a different model. **The outputs of the models are combined by voting to create a single output**. In real

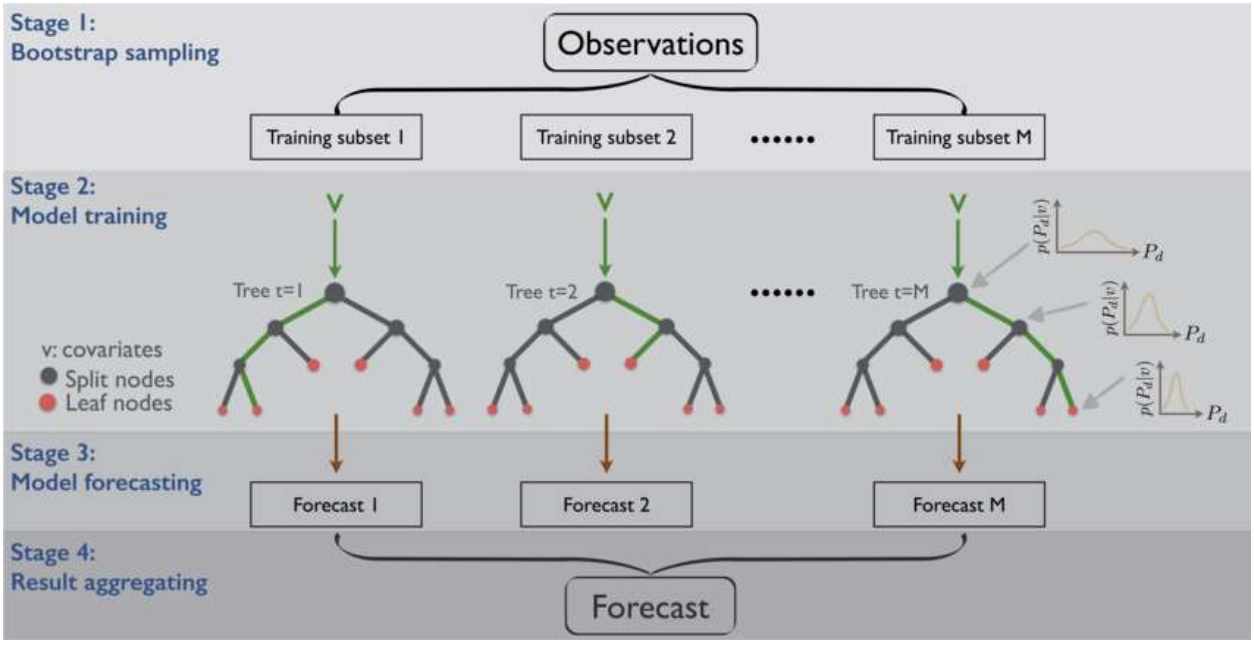
experiments, the bootstrapped samples are drawn from the training set, and the sub-models are tested using the testing set. The final output prediction is combined across the projections of all the sub-models.

The following infographic gives a brief idea of Bagging:



- In the bagging algorithm, the first step involves creating multiple models. These models are generated using the same algorithm with random sub-samples of the dataset which are drawn from the original dataset randomly with bootstrap sampling method.
- In bootstrap sampling, some original examples appear more than once and some original examples are not present in the sample. If you want to create a sub-dataset with m elements, you should select a random element from the original dataset m times. And if the goal is generating n dataset, you follow this step n times.





4.10 Boosting-based Ensemble learning (Converting Weak Models to Strong Ones):

Boosting, which was introduced by Schapire et al., is an ensemble method for boosting the performance of a set of **weak classifiers into a strong classifier**. This technique can be viewed as a model averaging method and it was originally

designed for classification, but it can **also be applied to regression**. Boosting provides **sequential learning of the predictors**. The first one learns from the whole data set, while the following learns from training sets based on the performance of the previous one. The misclassified examples are marked and their weights increased so they will have a higher probability of appearing in the training set of the next predictor. It results in different machines being specialized in predicting different areas of the dataset.

Boosting is a form of *sequential learning* technique. The algorithm works by training a model with the entire training set, and subsequent models are constructed by fitting the residual error values of the initial model. In this way, Boosting attempts to give higher weight to those observations that were poorly estimated by the previous model. Once the sequence of the models are created the predictions made by models are weighted by their accuracy scores and the results are combined to create a final estimation. Models that are typically used in Boosting technique are XGBoost (Extreme Gradient Boosting), GBM (Gradient Boosting Machine), ADABoost (Adaptive Boosting), etc.

Voting based Ensemble learning:

Voting is one of **the most straightforward Ensemble learning** techniques in which predictions from **multiple models are combined**. The method starts with creating two or more separate models with the same dataset. Then a Voting based Ensemble model can be used to **wrap the previous models and aggregate the predictions of those models.** After the Voting based Ensemble model is constructed, it can be used to make a prediction on new data. The predictions made by the sub-models can be assigned weights. *Stacked aggregation* is a technique which can be used to learn how to weight these predictions in the best possible way.

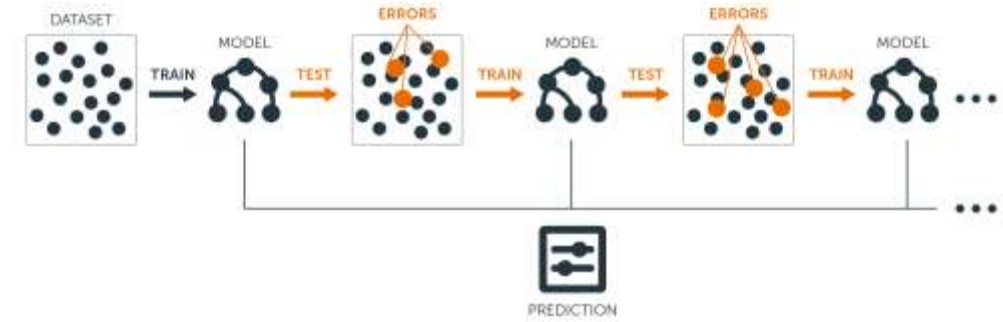
The following infographic best describes Voting-based Ensembles:

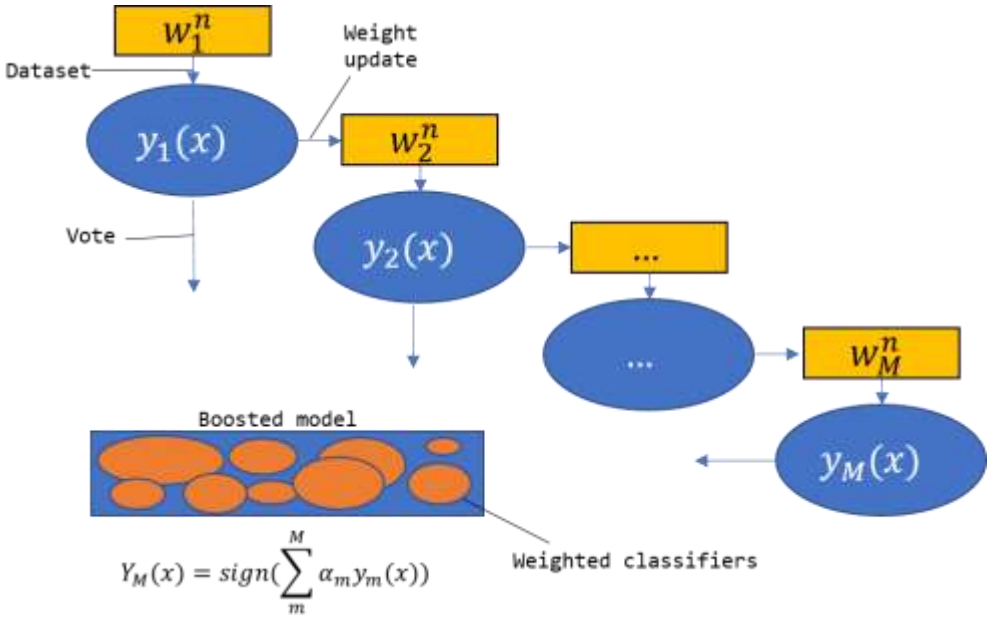
For example, if you have models **with high variance (they overfit your data)**, then you are likely to benefit from **using bagging**. If you have **biased models, it is better to combine them with Boosting**. There are also **different strategies to form ensembles**. The topic is just too broad to cover it in one answer.

But the point is: if you use the wrong ensemble method for your setting, you are not going to do better. For example, using Bagging with a biased model is not going to help.

Also, if **you need to work in a probabilistic setting, ensemble methods may not work either. It is known that Boosting (in its most popular forms like AdaBoost) delivers poor probability estimates**. That is, if you would like to have a model that allows you to reason about your data, not only classification, you might be better off with a graphical model. So, in this post, you got introduced to Ensemble learning technique. You covered its basics, how it improves your model's performance. You covered its three main types. Also, you implemented these three types in Python with the help of scikit-learn, and in this course of action, you gained a bit of knowledge about the necessary preprocessing steps.

- **Adaboost** is a widely known algorithm which is a boosting method. The founders of Adaboost won the Gödel Prize for their work. Mostly, decision tree algorithm is preferred as a base algorithm for Adaboost and in sklearn library the default base algorithm for Adaboost is decision tree (AdaBoostRegressor and AdaBoostClassifier). As we discussed in the previous paragraph, the same incremental method applies for Adaboost. Information gathered at each step of the AdaBoost algorithm about the ‘hardness’ of each training sample is fed into the model. The ‘adjusting dataset’ step is different from the one described above and the ‘combining models’ step is calculated by using weighted voting.

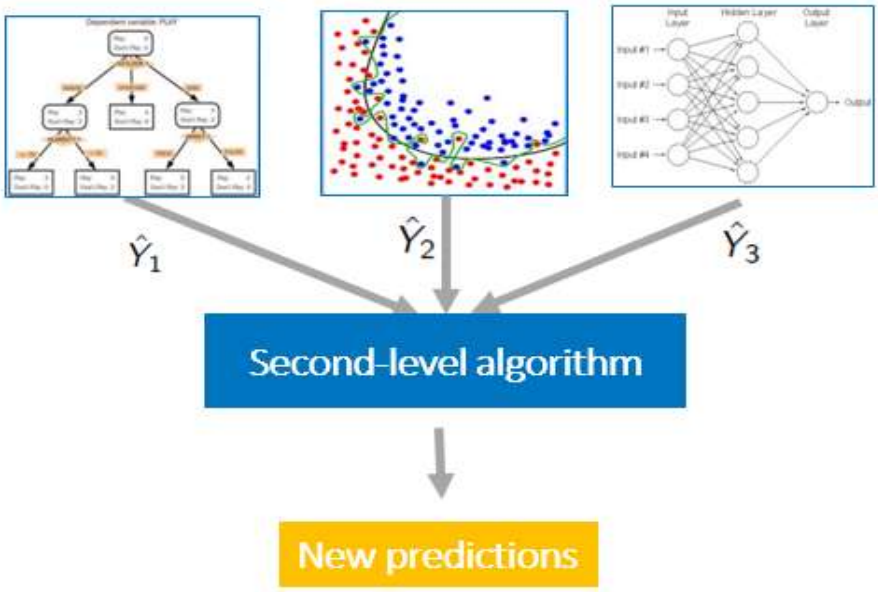




4.11 Stacking

- **Stacking**, also known as stacked generalization, is an ensemble method where the models are combined using another machine learning algorithm. The basic idea is to train machine learning algorithms with training dataset and then generate a new dataset with these models. Then this new dataset is used as input for the combiner machine learning algorithm
- **Model stacking** is an efficient ensemble method in which the predictions, generated by using various machine learning algorithms, are used as inputs in a second-layer learning algorithm. This second-layer algorithm is trained to optimally combine the model predictions to form a new set of predictions. For example, when linear regression is used as second-layer modeling, it estimates these weights by minimizing the least square errors. However, the second-layer modeling is not

restricted to only linear models; the relationship between the predictors can be more complex, opening the door to employing other machine learning algorithms.

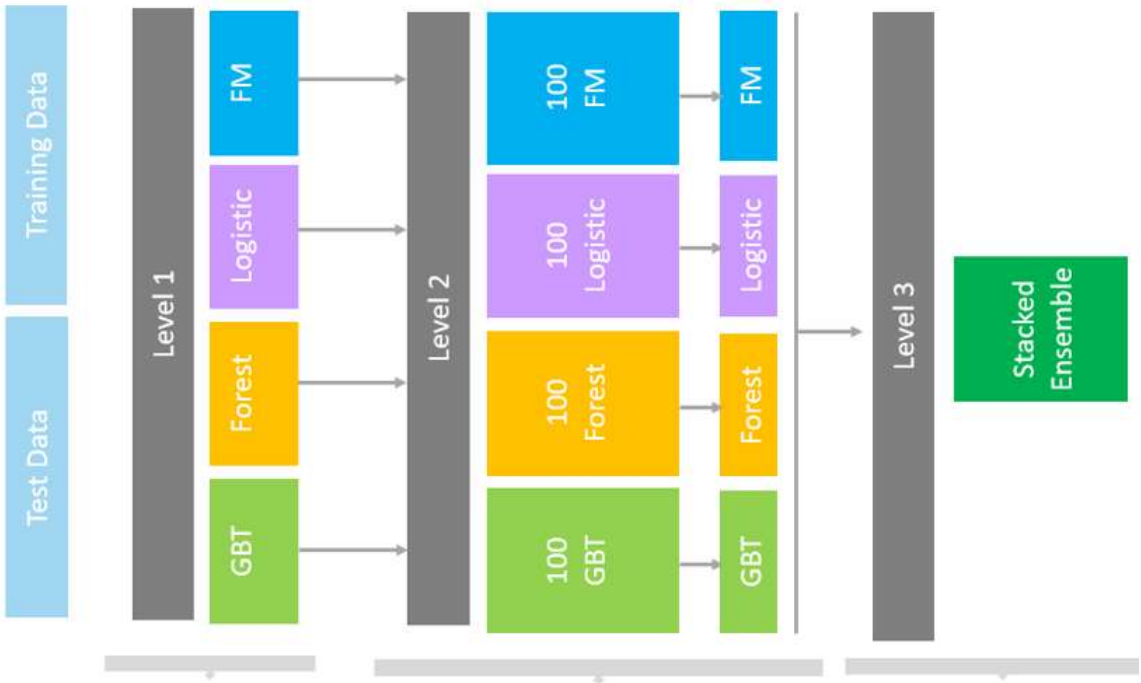


- Ensemble modeling and model stacking are especially popular in data science competitions, in which a sponsor posts a training set (which includes labels) and a test set (which does not include labels) and issues a global challenge to produce the best predictions of the test set for a specified performance criterion. The winning teams almost always use ensemble models instead of a single fine-tuned model. Often individual teams develop their own ensemble models in the early stages of the competition, and then join their forces in the later stages.
- The figure shows that a diverse set of 64 single models were used to build the model library. These models are trained by using various machine learning algorithms. For example, the green boxes represent gradient boosting models (GBM), pink boxes represent neural network models (NN), and orange boxes represent factorization machines models (FM). You can see that there are multiple gradient boosting models in the model library; they probably vary in their use of different hyperparameter settings and/or feature sets.
- At stage 1, the predictions from these 64 models are used as inputs to train 15 new models, again by using various machine learning algorithms. At stage 2 (ensemble

stacking), the predictions from the 15 stage 1 models are used as inputs to train two models by using gradient boosting and linear regression. At stage 3 ensemble stacking (the final stage), the predictions of the two models from stage 2 are used as inputs in a logistic regression (LR) model to form the final ensemble

•Jeong-Yoon Lee, Winning Data Science Competitions

- The most efficient techniques for training models (especially during the stacking stages) include using cross validation and some form of regularization



Automatic hyperparamter tuning

- Bootstrapping
- Training models and calculating 5-fold cross validated predictions
- Building four simple ensemble models

Stacking via

- Regularized regression
- Regularized scalable gradient boosting
- Hill climbing