**Model Ensemble**

Benefits :  
1. Imporoves stability and accuracy  
2. Reduces variances  
3. Avoids overfitting  
4. Compensates poor learning algorithms

**Examples** :

1. Bagging  
i. Binary estimates by model  
 a. Randomly create samples (with replacement)  
 b. create classifiers (same type!) and then run on test  
 c. use majority voting for classification on testing

ii. Probability estimates by model  
 a. Use the probability estimate and put in another classifier then on test  
2. Random Forest (bagging on DT learners) – promotes tree diversity   
3. Boosting (AdaBoost)  
 a. Seeks complementing models  
 b. Combines models of same type  
 c. new models handle errors of previous models  
 d. uses weighted averaging often adaptively  
4. Stacked Generalization (to combine multiple types of algos)  
 a. Different algos trained  
 b. Stacking (‘Level-1’) algorithm uses predictions from base (‘Level-0’) algorithms as inputs  
 i. Perform j-fold cross validation of training set (i.e. j different test sets each time)  
 ii. Train & test each of the Level-0 algos using split training data to create Level-0 models  
 iii. Test each model on each split to create Level-1 data  
 c. Crete multiple levels of stacking : Stacking of stacked classifiers  
5. Bayesian model combination : Built upon Bayes Model Averaging and Bayes Optimal Classifier  
 a. Bayes Optimal Classifier  
 i. Ensemble (using Bayes’ rule) of all hypothesis in hypothesis space  
 ii. Ideal emsemble  
 b. Bayes Model Averaging (BMA)  
 i. Approximates Bayes Optimal Classifier  
 ii. Samples from hypothesis space – monte carlo sampling  
 iii. Promotes overfit  
 c. Bayes Model Combination  
 i. Correction to BMA  
 ii. Uses model weightings to create samples – overcomes BMA drawbacks  
 iii. Better than BMA or bagging  
 Bayes optimal classifier maximizes the probability that the new instance is classified correctly given:  
- hypothesis space - a prior beliefs - observed data

Any of the following or related actions performed are considered as hyper-parameters for the Stacking :  
a. Not scaling the data  
b. Standard-Scaling the data  
c. Minmax scaling the data  
d. Feature selection  
e. Imputation type

**a) Voting Ensemble**

1. model ensembling reduces error rate
2. it works better to ensemble low-correlated model predictions.

Ensembles usually improve when adding more ensemble members, i.e, number of voters

Uncorrelated submissions clearly do better when ensembled than correlated submissions, and it results in an increase in the error-correcting capability.  
Weighted majority vote : Why weighing? Usually we want to give a better model more weight in a vote

**b) Averaging**

Averaging works well for a wide range of problems (both classification and regression) and metrics (AUC, squared error or logaritmic loss)  
- Averaging reduces overfit  
- Geometric mean can outperform a plain average.

**c) Rank Averaging**

First turn the predictions into ranks, then averaging these ranks. After normalizing the averaged ranks between 0 and 1 you are sure to get an even distribution in your predictions.

**d) Stacked Generalization**

It uses a pool of base classifiers, then using another classifier to combine their predictions, with the aim of reducing the generalization error (models on diffferent parts of train data are combined using their probabilities)

Let’s say you want to do 2-fold stacking:

* Split the train set in 2 parts: train\_a and train\_b
* Fit a first-stage model on train\_a and create predictions for train\_b
* Fit the same model on train\_b and create predictions for train\_a
* Finally fit the model on the entire train set and create predictions for the test set.
* Now train a second-stage stacker model on the probabilities from the first-stage model(s).

A stacker model gets more information on the problem space by using the first-stage predictions as features, than if it was trained in isolation.

Make sure that the level 0 generalizers are of all “types”, and not just simple variations of one another.Stacked generalization is a means of non-linearly combining generalizers to make a new generalizer, to try to optimally integrate what each of the original generalizers has to say about the learning set.’

**e) Blending (**or, Stacked Ensembling**)**

With blending, instead of creating out-of-fold predictions for the train set, you create a small holdout set of say 10% of the train set. The stacker model then trains on this holdout set only.

It is very close to in definition to Stacked Generalization, but a bit simpler and less risk of an information leak.

Blending has a few benefits:  
a) It is simpler than stacking.  
b) It wards against an information leak: The generalizers and stackers use different data.  
c) You do not need to share a seed for stratified folds with your teammates. Anyone can throw models in the ‘blender’ and the blender decides if it wants to keep that model or not.

The cons are:  
a) You use less data overall  
b) The final model may overfit to the holdout set.  
c) Your Cross-Validation is more solid with stacking (calculated over more folds) than using a single small holdout set.

**f) Stacking with Logistic Regression :** Use the logistic regression above the predictions of the individual models on the train and test dataset. Rather than model, one can go one step deeper and take the model predictions for each of the fold iteration for full test and train datasets, and then take the average of all the fold predictions for each of the model, and then put these input to the logistic model  
**g) Stacking with non-Linear algorithms** : Non-linear stacking with the original features on multiclass problems gives surprising gains. Obviously the first-stage predictions are very informative and get the highest feature importance. Non-linear algorithms find useful interactions between the original features and the meta-model features.

**h) Feature Weighted Linear Stacking**It stacks engineered meta-features together with model predictions. The hope is that the stacking model learns which base model is the best predictor for samples with a certain feature value. Linear algorithms are used to keep the resulting model fast and simple to inspect.   
Example : Vowpal Wabbit, it adds interactions between multiple feature namespaces (i.e., they can be engineered meta-features or original features only)

**i) Quadratic Linear Stacking of Models**   
Again in Vowpal Wabbit, using features as engineered ones (quadratic within a namespace or across namespaces) and then different combinations of weights to different model predictions.  
Example :

1 |f f\_1:0.55 f\_2:0.78 f\_3:7.9 |s RF:0.95 ET:0.97 GBM:0.92

This can easily be combined with feature-weighted linear stacking: -q fs -q ss, possibly improving on both.

Tune these base models to obtain model diversity. But at the end of the day you don’t know which base models will be helpful. And the final stage will likely be linear (which requires no tuning, or perhaps a single parameter to give some sparsity)

**j) Stacking classifiers with regressors and vice versa**Using classifiers for the regression problems and regressors for the classification problem in the final layer of stacking

**k) Stacking unsupervised learned features**Reduce the dataset to 2 or 3 dimensions using t-SNE and stack this with a non-linear stacker. Using a holdout set for stacking/blending feels like the safest choice here. Use t-SNE vectors and boost them with XGBoost.

**l) Online Stacking**first create small fully random trees from the hashed binary representation. Substract profit or add profit when the tree makes a correct prediction. Now take the most profitable and least profitable trees and add them to the feature representation. stack the XGBoost predictions together with the samples and let Vowpal Wabbit do what it does best: optimizing loss functions.