Predicting a biological response

With approach to Stacking and Weighted Ensemble

Note: Both Stacking and Weighted Ensemble are completely different approaches to problem. They have none in common. Scores for both Stacking and Weighted ensemble differ largely from each other.

CAVEAT

See the 4) Results 4.1)Model Evaluation and Validation. It would give sole idea of both techniques used(stacking and weighted ensemble models) and scores obtained for these two techniques.

To obtain best score, I avoided putting random_state in classfiers since as some more iterations are carried out over stacking technique(running same IPython NB 3-4 times) to obtain the max score.Same applies for Weighted Ensemble Model.

Predictions for both Stacking and Weighted Ensemble model are saved in submission.csv and submissionensemble.csv

Nupur Kamble

July 2017

1)Definition

1.1)Project Overview

Introduction.

People have used machine learning in various industries like pharma, robotics, computer vision, hedge funds etc with all sort of strategies from Bayesian statistics to optimization theories.

Scope of this project

This project aims at maximizing use of ensemble learning through stacking and weighted ensemble technique

Most of the kagglers are now using this ensemble techniques where ranks are calculated based on scores of cross validation scores and test set.

Why stacking and and weighted ensemble?

The kaggle has been house to various static datasets and offers almost real time raw data to contestants. However, since the competition relies on scoring technique like log loss and due to tremendous number of nearly log scores, it becomes important for kagglers to go for ensemble learning and stacking procedure.

The number one kagglers have demonstrated the power of ensembling and stacking in the model.

Aim of this project

The sole aim of this project is create a stacking and ensemble weighted models

So as to improve the accuracy score. The improve in score can be seen from weighted ensemble as we move towards stacking. However, introducing more complexity in weighted ensemble can surely give performance of model above that of stacking.

Data used in this project:

This is a classification problem.

Datasets and Inputs:

Features are chemical properties(D1 to D1776)

Target or Label is : Activity(0 and 1)(well balanced classes in target)

Inputs are: 1776 features from test set

Outputs: A submission file classifying each instance of test set as 0 and 1(Molecule ID)in terms of probabilities.Hence, log_loss score is calculated.

Training instance shape:(3751r,1777c)

All training features are quantitative in nature and straight.

Test set shape:(2501r,1776c)

No label for testing data. The testing labels and logloss is calculated on the competition link.We just submit predictions and logloss score is obtained for predictions from kaggle.

So, in order to select best models for stacking procedure, we apply GridSearchCV(validation and testing would be done) on

1)RandomForestClassifier
2)ExtraTreesClassifier
3)Gradient Boosting Model

To find best scoring model for each classifier by tuning hyperparameters.

Link: https://www.kaggle.com/c/bioresponse/data

1.2)Problem Statement:

1)This is a classification problem based on label 0 and 1.The label indicates the type of molecule we are going to predict based on 1776 columns or chemical properties.

The headers are kept anonymous so any sizeable judgement can't be made through seeing the features and labels.

2)Strategy: Since the dateset is purely quantitative in nature and large number of features are present, we could directly start for model generation. However, I have avoided using PCA to reduce the features since this can generate information loss and decided to work on raw data itself.

Methods used for solution:1)Stacking
2)Weighted Ensemble

a)Stacking: A general sense would be to construct meta features from collection of models that predict on (training ,crossvalidation sets) of its own in iterative manner.

So, we proceed by creating meta features from training set itself.

These meta features are used for training using a second learning algorithm. It can be

Support Vector Machines, Logistic Regression etc

We fit the model on meta features and predict for test set

b)Weighted Ensemble:

This is more mathematical approach. We use scipy optimize library and call the minimize function to find the suitable weights.

Approach: Collect predictions of models in a list.

Select suitable weights for every model such that misclassification error is minimized. The errors are reduced using scipy.optimize package.We import the minimize function from optimize package and then use it to find the weights of every model.

These weights are then multiplied to our list of predictions.

Hence, the weighted ensemble.

Detailed illustration of both Weighted Ensemble and Stacking procedure is in Algorithms and Methodology sections

1.3)Metrics

Metric used for ranking is log_loss.

This is the multi-class version of the Logarithmic Loss metric. Each observation is in one class and for each observation, you submit a predicted probability for each class. The metric is negative the log likelihood of the model that says each test observation is chosen independently from a distribution that places the submitted probability mass on the corresponding class, for each observation.

logloss=−1N∑i=1N∑j=1Myi,jlog(pi,j)logloss=−1N∑i=1N∑j=1Myi,jlog⁡(pi,j)

where N is the number of observations, M is the number of class labels, loglog is the natural logarithm, yi,jyi,j is 1 if observation ii is in class jj and 0 otherwise, and pi,jpi,j is the predicted probability that observation ii is in class jj.

The model that has been selected has a logloss of 0.37356 ranking first in 729 competitiors.This is my benchmarking model.
Source: Kaggle leaderboard: https://www.kaggle.com/c/bioresponse/leaderboard

My submission produces a logloss score of 0.37751
This logloss score ranks around top 16 members out of 699 teams with 792 competitors.
1.4)Evaluation Metrics

Using log_loss for evaluation of model
We import log_loss from sklearn as a metric our competitions choice.
sklearn.metrics.**log_loss**(*y_true*, *y_pred*, *eps=1e-15*, *normalize=True*, *sample_weight=None*)

Log loss, aka logistic loss or cross-entropy.

This is the loss function used in (multinomial) logistic regression and extensions of it such as neural networks, defined as the negative log-likelihood of true labels given a probabilistic classifier's predictions. For a single sample with true label yt in {0,1} and estimated probability yp that yt = 1, the log loss is
-log P(yt|yp) = -(yt log(yp) + (1 - yt) log(1 - yp))

2)Analysis

2.1)Data Exploration:

The data is purely quantitative in nature:

Since, we can search for skewness of data.

Examining data statistics of training: Seeing the notebook we can certainly see that no negative values are present for checking skew.

Thus , we check for skew values of data as:data.values.skew()

Which shows some high skewness in columns. In order to reduce the skew, lets apply log to entire dataframe. However, even after applying np.log to entire dataframe and checking skew values, there is no significant reduction in skew of some columns.

Thus, I decided to directly go for training the original unskewed dataset.

Note: Checking Outliers:

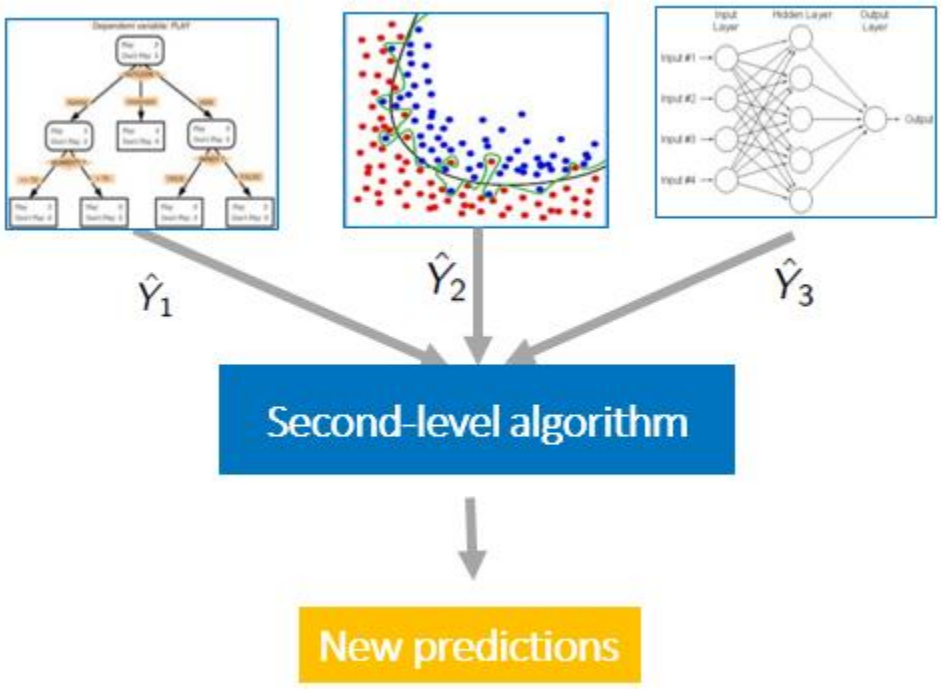
I have decided not to go for checking outliers:

Possible reasons: Since, this dataset contains only types of molecules for every 1776 properties, it sounds pretty naïve to calculate outliers in this dataset.

Plotting Correlation map using seaborn of reduced dimensions:

No high correlation found between features variables. Thus, we can move directly to model building process.Heatmap using seaborn has been plotted.

2.2)Exploratory Visualization:



The above picture clearly depicts the process of stacking. Lets breakdown the picture into following steps to understand the process.

1)Divide the data into 10 equal parts. Each piece(single) of data acts as a test set(validation) and remaining 9 pieces forms the training set i.e. train model on 9 parts of data and predict on remaining part.

The above process is repeated by predicting every single piece of data and training over remaining 9 parts of data.

2)3 classifiers are taken and for each classifier we generate the predictions for complete data by following step 1. i.e. 9 parts for fitting the model and remaining for training.

Simultaneously, predictions are also made over real testing set, and mean of predictions are taken after completing 10 iterations and saved in blend_test.

3)So, we have a new prediction set made over training data. They are called meta feature(blend_train) . Now, use these meta features for training and predicting over blend_test using a meta classifier(Second_level_algorithm). In notebook, I have used Logistic Regression as meta classifier.

Hence, the above diagram clearly represents the process.

2.3)Algorithms and Techniques

Chosen Classifiers:

A)RandomForestClassifier: **Random forests** or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees habit of overfitting to their training set.

1. Randomly select **"k"** features from total **"m"** features.
    1. Where **k << m**
2. Among the **"k"** features, calculate the node **"d"** using the best split point.
3. Split the node into **daughter nodes** using the **best split**.
4. Repeat **1 to 3** steps until "l" number of nodes has been reached.
5. Build forest by repeating steps **1 to 4** for "n" number times to create **"n" number of trees**.

B)ExtraTreesClassifier

An "extra trees" classifier, otherwise known as an "Extremely randomized trees" classifier, is a variant of a random forest. Unlike a random forest, at each step the entire sample is used and decision boundaries are picked at random, rather than the best one. In real world cases, performance is comparable to an ordinary random forest, sometimes a bit better.

C)GradientBoostClassifier

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function

The following link clearly explains the GradientBoost:

http://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html

For Layman terms, the following link helps so much to understand

http://blog.kaggle.com/2017/01/23/a-kaggle-master-explains-gradient-boosting

Stacking

Process is as follows: In order to select the best algorithms for stacking we perform GridSearchCV over the RandomForestClassifier, ExtraTreesClassifier. GridSearchCV over GradientBoostingClassifier is avoided due to time consuming process which almost takes half an hour to find the best parameters.

However, the scores don't differ much with manual and best selected parameters from GridSearchCV.

The input to algorithms is training dataset excluding response or label (Activity).

They are then converted to numpy array using dataframe.values for ease with model fitting.

Datasets and Inputs:
Features are chemical properties(D1 to D1776)
Target or Label is : Activity(0 and 1)(well balanced classes in target)
Inputs are: 1776 features from test set
Outputs: A submission file classifying each instance of test set as 0 and 1(Molecule ID)in terms of probabilities.Hence, log_loss score is calculated.
Training instance shape:(3751r,1777c)
All training features are quantitative in nature and straight.
Test set shape:(2501r,1776c)
No label for testing data. The testing labels and logloss is calculated on the competition link.We just submit predictions and logloss score is obtained for predictions from kaggle.

Both RF and GBM are ensemble methods, meaning you build a classifier out a big number of smaller classifiers. Now the fundamental difference lies on the method used:

1. RF uses decision trees, which are very prone to overfitting. In order to achieve higher accuracy, RF decides to create a large number of them based on bagging. The basic idea is to resample the data over and over and for each sample train a new classifier. Different classifiers overfit the data in a different way, and through voting those differences are averaged out.
2. GBM is a boosting method, which builds on weak classifiers. The idea is to add a classifier at a time, so that the next classifier is trained to improve the already trained ensemble. Notice that for RF each iteration the classifier is trained independently from the rest.

Stacking Theory:

Stacking (also called meta ensembling) is a model ensembling technique used to combine information from multiple predictive models to generate a new model. Often times the stacked model (also called 2nd-level model) will outperform each of the individual models due its smoothing nature and ability to highlight each base model where it performs best and discredit each base model where it performs poorly. For this reason, stacking is most effective when the base models are significantly different.

Weighted Ensemble:

We are using raw approach for weighted ensemble. We calculate the predictions of each model and then save it to the prediction list. We import optimize package from scipy and use minimize function in order fit the model as

Minimize(W1x1 +W2x2+….+Wnxn) such that log loss of is also minimized. Constraints are also applied such as weights should be between 0 and 1. Also, they must sum upto 1.

The minimizing algorithms used is SLSQP(Sequential least Squares Programming)

https://docs.scipy.org/doc/scipy/reference/optimize.minimize-slsqp.html

Minimize function returns suitable weights such that log loss is minimized.

2.4) Benchmark

Metric used for ranking is log_loss.

This is the multi-class version of the Logarithmic Loss metric. Each observation is in one class and for each observation, you submit a predicted probability for each class. The metric is negative the log likelihood of the model that says each test observation is chosen independently from a distribution that places the submitted probability mass on the corresponding class, for each observation.

logloss=−1N∑i=1N∑j=1Myi,jlog(pi,j)logloss=−1N∑i=1N∑j=1Myi,jlog⁡(pi,j)

where N is the number of observations, M is the number of class labels, loglog is the natural logarithm, yi,jyi,j is 1 if observation ii is in class jj and 0 otherwise, and pi,jpi,j is the predicted probability that observation ii is in class jj.

The model that has been selected has a logloss of 0.37356 ranking first in 729 competitiors.This is my benchmarking model.
Source: Kaggle leaderboard: https://www.kaggle.com/c/bioresponse/leaderboard

My submission produces a logloss score of 0.37751
If more iterations are carried out a log loss score of 0.37564 can be obtained ranking 7th in Private leaderboard
This logloss score ranks around top 16 members out of 699 teams with 792 competitors.
Top rank has logloss of 0.37356 on private leaderboard

https://www.kaggle.com/c/bioresponse/leaderboard

3)Methodology

3.1)Data Preprocessing

a)The data doesn't have missing values and Nan. All the data is quantitative in nature and none of them is categorical except the classification label Activity.

b)Observing the skewness of data using data.skew(), it can be observed that most of quantitative variables are between -1 to 1 and some are just close. Apply np.log to features won't generate much leverage.

c)The data is scaled. No large gap between the values of different features are seen.

Hence, the data provided is clean.

PCA can be used since 1776 features are present. However, it is better to use all the features for training model.

For checking the correlation after transforming features to 4 components, it is found that there is almost no correlation after plotting the heatmap between the features using seaborn.

3.2)Implementation

Here is the flow of stacking algorithm

X = Features set , y=response and test_set= test

blend_train: np.zeros((X.shape[0],len(classifiers))

blend_test:np.zeros(test.shape[0],len(classifiers))

1)Generate crossvalidation indices using skf=StratifiedShuffleSplit for k=10

2)for i, classifier in enum(classifierlist)

blend_test_j = np.zeros(test.shape[0],len(skf))

For j,(trainind, testind) in skf:

```
        Xtrain = X[trainind]
        Ytrain = y[trainind]
        Xtest = X[testind]
        Ytest = y[testind]
        Clf.fit(xtrain,ytrain)
        blend_train[:,i]=clf.predict_proba(xtest)[:,1]
        blend_test[:,j] = clf.predict_proba(test)[:,1]
    blend_test[:,i] = blend_test_j.mean(1)#mean along x axis
```

We have generated meta features. Now, use another classifier to train on (blend_train, Y) and predict on blend_test

clf = LogisticRegression().fit(blend_train,Y).predict_proba(blend_test)[:,1]

Thus, we classified the molecules using stacking algorithm

We save the predictions

Problems and some dropouts I faced in here was:To select a diverse set of RandomForest, ExtraTrees and GradientBoostClassifier. Earlier, I thought base models like SVM, Decision trees, Logistic Regression but from searching some texts online and given the nature of competition which is to maximize the accuracy, it made sense to go for RandomForest, ExtraTree and GradientBoost classifiers.

Also, I wasn't able to decide whether predictions on test set should be made on basis of)Majority Voting2)Mean3)Plurality. However, getting the mean is the standard procedure to follow during such competitions.

Code Snippet:

```
blend_test[:,i] = blend_test_j.mean(1)#mean along x axis
```

Weighted Ensemble Model

Algorithm: 1) Save the predictions of each classifier as

predictions =[p1,p2,p3,p4]

2) Now, import minimize function from scipy.optimize

3)Set the constraints: a)starting values(weights) =

0.5 weight for every prediction

b)weights are bound between 0 and 1

c)con=({'type':'eq','fun':lambda w: 1-sum(w)})

d) Select the SLSQP method for minimizing the log_loss_func which takes weights as numpy array.

4)run: res = minimize(log_loss_func, starting_values, method='SLSQP', bounds=bounds, constraints=cons).

5)extracts weighs from res as weights = res['w']

6)now,preds= weights[0]*predictions[0]+weights[1]*predictions[1]+ weights[2]*predictions[2]+weights[3]*predictions[3]

7)Submit these preds to log_loss function in order to calculate log_loss

Hence, the weighted ensemble algorithm.

Caveats and Problems during implementation:

Since, I had found the weights of model to combine. I was a little bit confused whether I should go towards a stacking approach or directly combine the models to generate predictions.

Final predictions =
weights[0]*predictions[0]+weights[1]*predictions[1]+weights[2]*predictions[2]+weights[3]*predictions[3]

But making direct predictions using model weights didn't work much efficient than the stacking model. This proves the power of stacking.

Also, there were some scripts which were really complex while making predictions using model weights. A script that I tried to understand but was way too far from simplicity is

https://www.kaggle.com/svpons/three-level-classification-architecture

I have become so obsessed with the above script. It's a more complex implementation using model weights.

3.3)Refinement

a)I have used GradientBoostingClassifer, RandomForestClassifier and ExtraTreesClassifier. Finding the parameters for tuning was a trial and error method. However sklearn provides a much better approach such as GridSearchCV. But in case of ensemble, it is advisable to use the multiple models with not so much difference in their metrics. So, I decided to add 2 different RandomForestClassifier and ExtraTreesClassifier along with GradientBoostClassifier.

I haven't used GridSearchCV here since I selected multiple models of same type for stacking procedure.

Note: The score produced must be nearby each other as (0.472,0.473,0.446,0.45) as opposed to (0.472,0.553,0.50,0.43) which shows a lot of difference. Ensembling models with a lot of difference in scores can bring down the OVERALL score. Hence, it is advised to keep those models which produce a less difference in their respective scores. Thus overall score can be much higher than the scores obtained from models.

b) In Weighted Ensemble implementation, I started out with ExtraTreesClassifier,RandomForestClassifier and Gradient boost Classifier. But, logloss achieved varied widely on all the four classifiers. So, again a ensemble of 2 RandomForestClassifier and 2 models of GradientBoostClassifier gave a really nice logloss score which didn't varied widely.

The log loss scores are as follows

rfc1 classifier score :0.472675007385
rfc2 classifier score :0.473259927306
gbc1 classifier score :0.446371041036
gbc2 classifier score :0.453345761703

```
sss = StratifiedShuffleSplit(labels, test_size=0.05, random_state=1234)
for train_index, test_index in sss:
    break
train_x, train_y = train[train_index], labels[train_index]
test_x, test_y = train[test_index], labels[test_index]
```

The log _loss scores are obtained by training on training set and predicting over test sets obtained from Stratifiedsplit in the above code snippet.

4)Results:

4.1)Model Evaluation and Validation:

I used two different models which generated two different scores on kaggle.First model is Stacking and second one is Weighted ensemble. Stacking produces the best score on kaggle leaderboard while Weighted ensemble doesn't perform well. The predictions of both of these models are saved in submission.csv and submissionensemble.csv

The model that has been selected has a logloss of 0.37356 ranking first in 729 competitiors.This is my benchmarking model.
Source: Kaggle leaderboard: https://www.kaggle.com/c/bioresponse/leaderboard

My submission(Stacking Model with predictions in submission.csv) produces a logloss score of 0.37559,0.37564 and 0.37770 in 3 different iterations(running the stacking model 3 times and selecting the best score submission file name: submission.csv
And in case of Weighted Ensemble scores in 2 iterations are: 0.39658 and 0.41342 which gives a rank around 150.submission file name:submissionensemble.csv
This logloss 0.37770 score ranks around top 16 members out of 699 teams with 792 competitors while log loss of 0.37559 and 0.37564 ranks 7th
Also, when the stacking model is run over several iterations, more improvement can be found.My maximum score is 0.37564 which ranks 7^{th} in competition.
The model is derived by performing gridsearch over randomforestclassifier, gradientboostclassifier and extratreesclassifier as these algorithms are favorites among kagglers. An ensemble of these models was used for stacking.

Validation can be done solely on basis of log_loss score kept as a metric for competition. Thus, top ranker had a log_loss of 0.37356. All, the details regarding private leaderboard ranking can be found here https://www.kaggle.com/c/bioresponse/leaderboard

4.2)Justification

Benchmark: Top rank model which produces a log_loss score of 0.37356 has been selected as a benchmark.

My solution is based on increasing the accuracy or decreasing the log loss so as to get into top ranks of competition. It's a bit difficult to to enter top ranks without ensembles in kaggle competition.

The analysis is solely based on log_loss score which is 0.37556 which was obtained after several iterations.

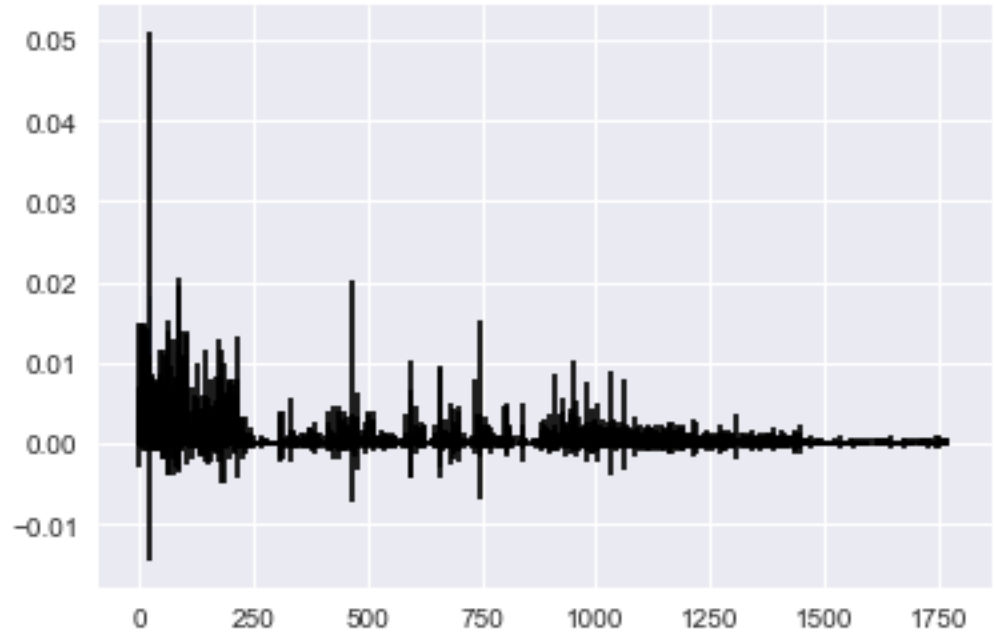
The model obtained is surely a well performing model.

I think this model is significant enough to solve the problem.

5)Conclusion

5.1)Visual Exploration

Exploration of feature importances of all columns 1776 of dataset.



Lets plot the feature importances and analyze it:

Since, there are almost 1776 features in training set, it becomes much more important to find most important features that could help in preparing a good model. From the graph it can be observed that features ranging from upto 250 show a strong importance and then onwards it goes down.

However, it must be noted that reduction in dimensionality accompanies loss of information. Dimensionality reduction should be used where time consumption matters. However, in this kaggle dataset, it wasn't time consuming for modeling of data.

The data is straightforward and quantitative in nature. Also, given the nature of problem with no test set labels, it quite difficult to visualize the ensemble learner.The labels for test set at kaggle site isn't revealed. Hence, it is not possible to plot any kind of learning curves, error graphs etc.

It's a more mathematical approach.

Hence, I would like to express key points here instead of visualization.

They average out biases

If you average a bunch of democratic-leaning polls and a bunch of republican-leaning polls together, you will get on average something that isn't leaning either way

They reduce the variance

The aggregate opinion of a bunch of models is less noisy than the single opinion of one of the models. In finance this is called diversification - a mixed portfolio of many stocks will be much less variable than just one of the stocks alone. This is also why your models will be better with more data points rather than fewer.

They're unlikely to overfit

If you have individual models that didn't overfit, and you're combining the predictions from each model in a simple way (average, weighted average, or logistic regression), then there's no room for overfitting.

Some models such as **forest models** are already ensemble models.

5.2)Reflection

So, in final phase I would like to summarize both the algorithms for model building:

1)Stacking Model: This model constructs meta features from cross validation sets and used to fit the next classifier and make prediction on meta test set.

Flow:

A)Construct meta features i.e blend_train which contains predictions over training set(meta features) using cross validation.In same way, take predictions over test set and take mean of every row of prediction.

B) Now, take another classifier or meta classifer which uses blend_train for fitting the model and predicting over blend_test. i.e. clf.fit(blend_train,y) and then predict over clf.predict(blend_test)

Some Interesting intuition: We can use two meta classifiers and find the weights of each classifier in order to improve accuracy. Also, instead of GradientBoost I would suggest to use Xgboost since it is much faster and suitable for such competitions.

If you have models with high variance (they over-fit your data), then you are likely to benefit from using bagging. If you have biased models, it is better to combine use them with Boosting. There are also different strategies to form ensembles. The topic is just too wide to cover it in one answer.

But my 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.

2)Weighted Ensemble Model:

This model is based on mathematics i.e we minimize the log loss and find suitable weights for combination of models. However, I found that just finding weights and predicting won't give much better results. Weighted Ensemble Model needs a lot of feature engineering and a bit complicated meta classifiers.

Optimization is also an issue with weighted ensemble model. There are several methods in scipy to find the weights. We have used the SLSQP method for finding the weights. However, there are even better methods that can be used to optimize the weights. Once such method is

L-BFGS-B for minimizing logloss. Also, we can combine both optimization methods and find suitable weights.

Here is the flow of stacking algorithm

X = Features set , y=response and test_set= test

blend_train: np.zeros((X.shape[0],len(classifiers))

blend_test:np.zeros(test.shape[0],len(classifiers))

1)Generate crossvalidation indices using skf=StratifiedShuffleSplit for k=10

2)for i, classifier in enum(classifierlist)

blend_test_j = np.zeros(test.shape[0],len(skf))

For j,(trainind, testind) in skf:

Xtrain = X[trainind]

Ytrain = y[trainind]

Xtest = X[testind]

Ytest = y[testind]

Clf.fit(xtrain,ytrain)

blend_train[:,i]=clf.predict_proba(xtest)[:,1]

blend_test[:,j] = clf.predict_proba(test)[:,1]

blend_test[:,i] = blend_test_j.mean(1)#mean along x axis

We have generated meta features. Now, use another classifier to train on (blend_train, Y) and predict on blend_test

clf = LogisticRegression().fit(blend_train,Y).predict_proba(blend_test)[:,1]

Thus, we classified the molecules using stacking algorithm

We save the predictions

Weighted Ensemble Model

Algorithm: 1) Save the predictions of each classifier as

predictions =[p1,p2,p3,p4]

2) Now, import minimize function from scipy.optimize

3)Set the constraints: a)starting values(weights) =

0.5 weight for every prediction

b)weights are bound between 0

and 1

c)con=({'type':'eq','fun':lambda w: 1- sum(w)})

d) Select the SLSQP method for

minimizing the log_loss_func

which takes weights as numpy

array.

4)run: res = minimize(log_loss_func, starting_values, method='SLSQP', bounds=bounds, constraints=cons).

5)extracts weighs from res as weights = res[‘w’]

6)now,preds= weights[0]*predictions[0]+weights[1]*predictions[1]+

weights[2]*predictions[2]+weights[3]*predictions[3]

7)Submit these preds to log_loss function in order to calculate log_loss

Hence, the weighted ensemble algorithm.

5.3)Improvement

Improvement in Stacking:

I would like to add some more diverse classifiers with more diverse parameters such as splits for random_forest_classifier, learning rate for Gradientboostclassifier and max depth for extra trees classifier. However, the trade off is time consumption. GradientBoost really takes a lot of time to train with different learning parameters.

Moreover, we can use xgboost instead of logistic regression for predicting meta features of ensemble. Support Vector Classifier is also a good choice but in case of increasing accuracy, xgboost would certainly help

Improvement in Weighted Ensemble:

I have implemented a straightforward ensemble where in weights are directly assigned to predicted probabilities of different classifiers and minimized using optimize package of scipy.

A more performance increasing approach is to use stacking and use two different models such as xgboost and gradientboostingclassifier to make predictions.

We can then find weights that minimize logloss thus improving accuracy of model.