Idan Yamin

IMDB data set

The data set I have used is available at https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews
This data set contains 50,000 reviews from IMDB, 50% of them are good reviews and 50% are bad reviews.
Each review is labeled positive or negative.

User's guide to the program can be found in page 11.

Goal: Build a classifier that when given a review states whether it's a positive or negative review.

Preprocessing:
I have divided the data set into 3 sections: data vault, train and test.

Train: The training data is used to train the classifier.

Test: The testing data is a set of reviews and labels that suppose to give a feedback whether the classifier works well on samples it didn't train upon.
We need the test data in order to check variance and overfitting.

Data vault: The data vault purpose is after finding the best classifier, it gives us a feedback whether we overfitted over the training and testing data. We should expect the error on the vault to be like the error on the testing data. If it's not the case, we should consider the probability that our classifier overfitted over the testing data.

After dividing the data into 3 sections, I created a bag of words.
The bag is composed of words that appeared in the training data set.
Each review is than translated to its bag of words representation.
Example: bag of words = {movie, good, bad}
The review 'The movie was good' will be translated to a vector
of the form [1,1,0] because 'movie' appeared once, 'good' appeared once and 'bad' didn't appear in the sentence at all.
Notice we can write a review that can be confusing like.
'I have nothing good to say about the movie'
Like the sentence above the vector representing it is [1,1,0]
It has the same representation as a vector of a good review because it has the word 'good' in it. That's why we should be careful of how we choose the words in the bag.

Baseline:
I took a classifier that succeed poorly on the testing data to gain a baseline.
The baseline classifier job is to give us a bound from below of the error rate.

For the baseline I have used logistic regression.
I trained the logistic regression on all the words in the bag of words:
The mean 0-1 loss on the test data is: $\frac{1}{n}\Sigma_{i=1}^{n}|y_i - predict(x_i)| = 0.11205387205387206$.
n = number of samples in the test set.

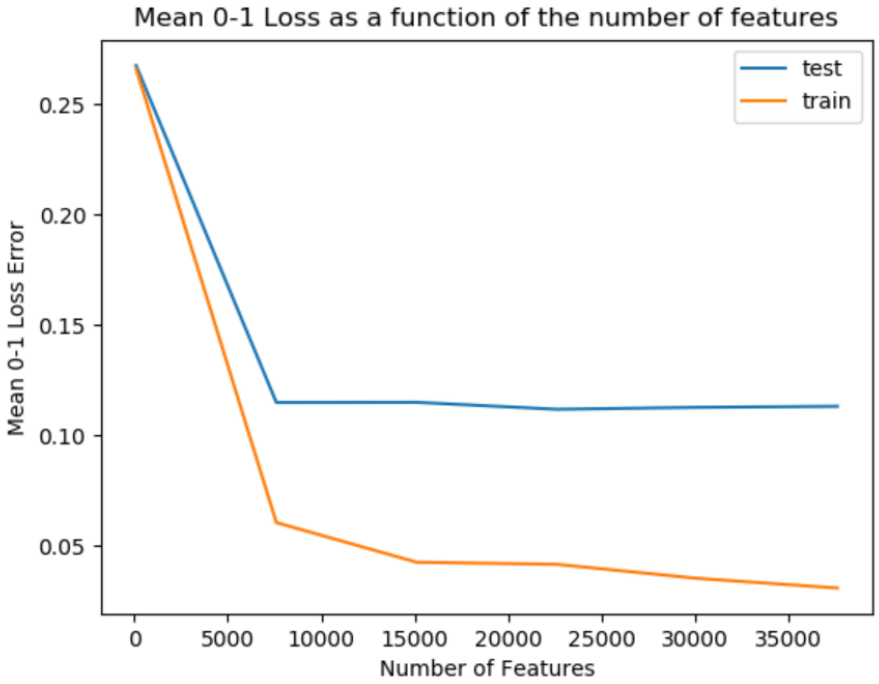
Idan Yamin

y = true label of review (1=positive, 0=negative)
predict = prediction rule in our case logistic regression.
x = review as a bag of word representation.

In general, when using logistic regression having more features than data is bad.
It may cause high variance meaning larger error on the testing sample.

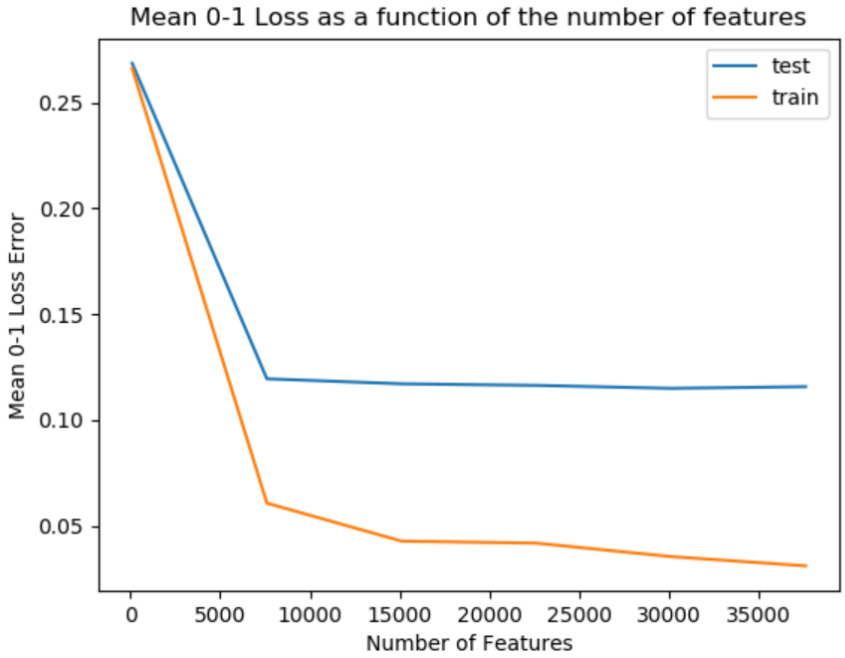
I tried to figure out the minimum features that are required for the logistic regression to work well.
I came up with a method of feature selection.
I took the X most commonly used words in the bag of words and trained the classifier just on those words, and then plotted the mean 0-1 error.
Because it is computationally hard problem, I took just 6 points and plotted them.

Here in graph, we can see the mean 0-1 error as a function of the number of features.
The blue line is the error on the test.
The orange line is the error on the train.
We can see that when increasing the number of most commonly used features (words) the mean error starts to converge around 0.12



Idan Yamin

This graph is the results using cross-validation method, the data was divided into 10 folds.
It looks almost identical to graph above.



Conclusion, the logistic regression needs approximately 17,500 features to converge to the optimal solution.

Now that we have achieved the baseline classifier, we can get rid of any classifier that does worse than that.

An example of a bad classifier

Let's look at an example of a bad classifier.
I applied linear regression on 17,500 features.
The mean 0-1 loss is 0.4057107053835907 it's much worse than the baseline classifier results. It emphasizes the importance of a baseline classifier.
Let's try to take that example and apply regularization on it.

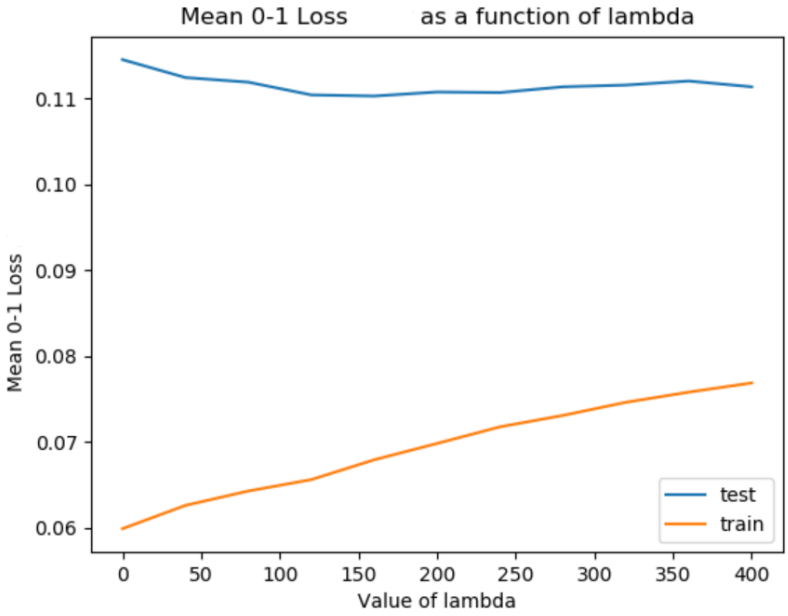
Ridge regression

In general ridge regression reduces variance and model complexity because it shrinks the coefficient. In the ridge case we are looking for:

$argmin_w \left||y - wx|\right|_2^2 + \lambda\left||w|\right|_2^2 \quad (y = true\ label, x = samples, w = weights)$

In order the find the optimal lambda we should use the test data/cross validation.
Now let's try to plot the mean 0-1 loss as a function of λ.

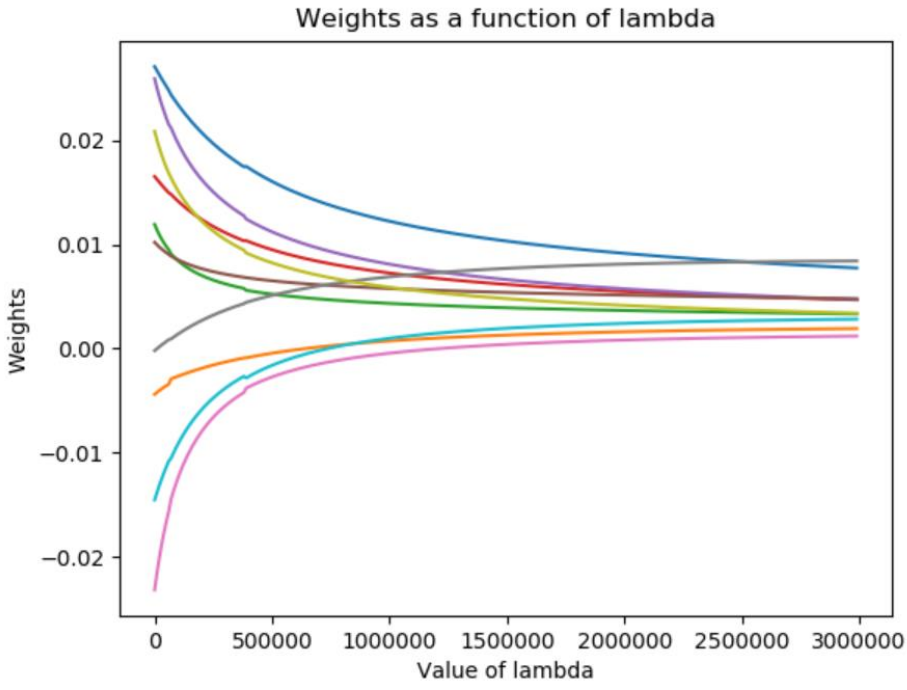
Idan Yamin



As expected, when lambda increases the training error also increases but there is a slight improvement in the test error. We reduced the variance but increased the bias.

As we can see when lambda is around 125, we get the best results and it's slightly better than the baseline classifier, and much better than linear regression.

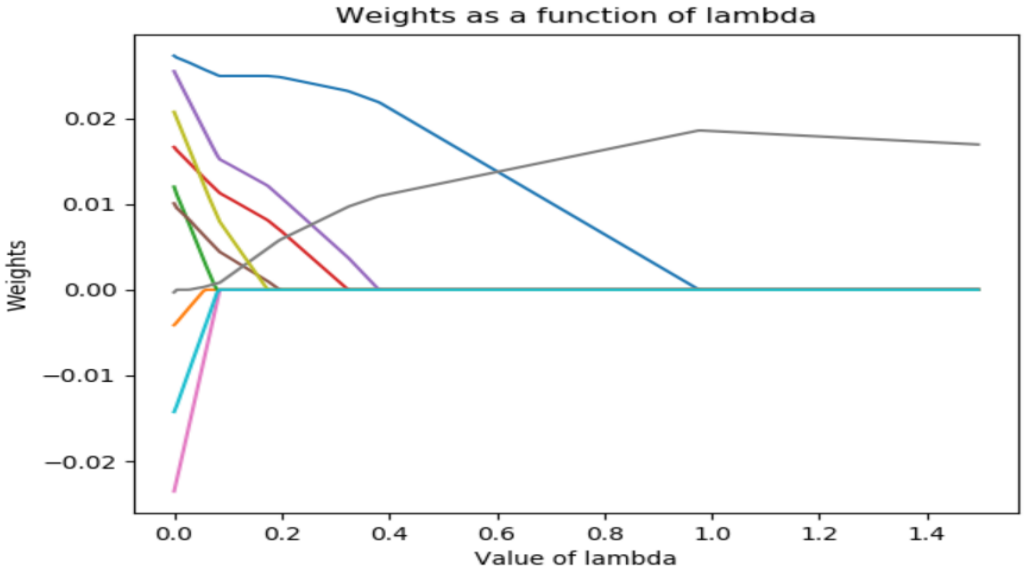
Let's look at the regularization path. I used only 10 coefficients for the graph. Large amount would cause the graph to be unreadable.



as we can as lambda goes to infinity the coefficients of w go to zero.

Lasso Regression

Lasso tends to make coefficients that are noisy zero.
In the Lasso case we are looking for:
$argmin_w ||y - WX||_2^2 + \lambda ||W||_1^2$
Now let's plot the regularization path.



As we can see the regularization path of Lasso tends to converge quicker than Ridge, also it tends to send coefficients to zero, it works well when some coefficients are irrelevant for the final prediction.

In order to decrease computation time I could have used PCA (Principal Component Analysis) to reduce the dimensionally of the data.
Meaning finding $argmin_{W,U} ||X - UWX||$. (X is the data matrix, W is the dimension reduction matrix and U is the reconstruction matrix) Intuitively minimizing that term meaning we lost minimal data applying linear data reduction and reconstruction.
But we saw that the classifiers so far didn't do well. applying PCA would likely increase the generalization error.

Until now we only used linear classifier in general most problems don't tend to be linearly separable, in some cases we should apply kernel function and then use linear classifier like soft SVM (Support Vector Machine), the tough problem here is to find a function that when applying it on the data will separate the data linearly.

Idan Yamin

Adaboost

The algorithm:

- **input:** training set $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$, weak learner WL, number of rounds T
- **initialize** $\mathbf{D}^{(1)} = (\frac{1}{m}, \ldots, \frac{1}{m})$
- **for** $t = 1, \ldots, T$:
  - invoke weak learner $h_t = \mathrm{WL}(\mathbf{D}^{(t)}, S)$
  - compute $\epsilon_t = L_{\mathbf{D}^{(t)}}(h_t) = \sum_{i=1}^{m} D_i^{(t)} \mathbb{1}_{[y_i \neq h_t(\mathbf{x}_i)]}$
  - let $w_t = \frac{1}{2} \log\left(\frac{1}{\epsilon_t} - 1\right)$
  - update $D_i^{(t+1)} = \frac{D_i^{(t)} \exp(-w_t y_i h_t(\mathbf{x}_i))}{\sum_{j=1}^{m} D_j^{(t)} \exp(-w_t y_j h_t(\mathbf{x}_j))}$ for all $i = 1, \ldots, m$
- **output** the hypothesis $h_s(\mathbf{x}) = \mathrm{sign}\left(\sum_{t=1}^{T} w_t h_t(\mathbf{x})\right)$.

Intuitively, the weak learner on each iteration focuses on the samples it failed on.
And after T iteration it returns a linear combination of weak learners to gain a strong prediction rule.

In order to run the Adaboost algorithm we firstly need to choose a weak learner.
Let's try a decision tree with 10 leaves at most.

A weak learner must be at least a bit stronger prediction rule than a random classifier.
In our case a random classifier will approximately succeed on 50% of the test data.
The decision tree with at most 10 leaves has a mean 0-1 loss of 0.2952188552188552.
It's better than a random coin flip.
Now I will use Adaboost on a decision tree with at most 10 leaves.
The Adaboost will do 150 iteration until it would stop and return a prediction rule.
The mean 0-1 error of Adaboost is: 0.12228956228956234
The results are not as good as the baseline classifier, meaning we should try another approach.

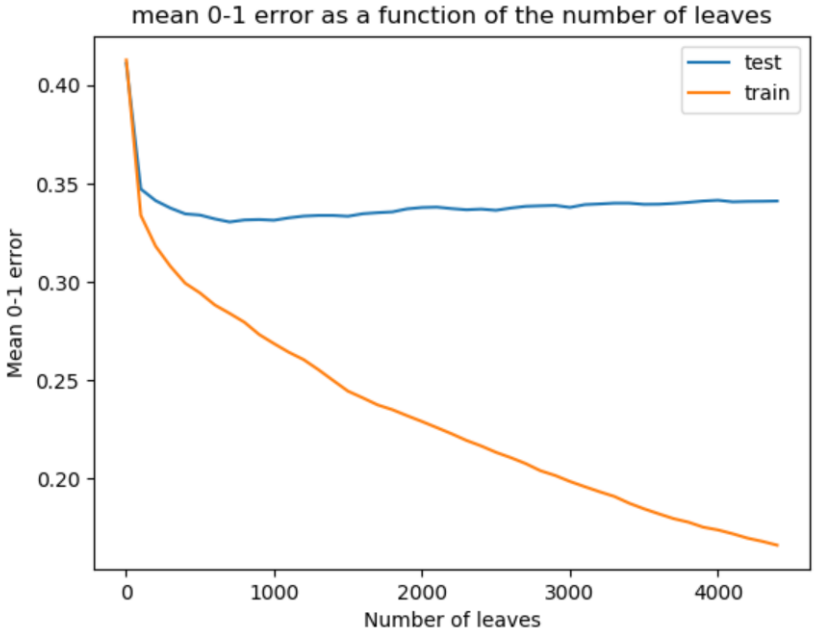
Idan Yamin

Random forest

In general decision tress tend to overfit the training data (low bias, high variance).
Random forest is a way of averaging multiple decision trees.
Each tree is trained on different part of the same training set.
the goal is to reduce variance. Random forest tends to increase a bit the bias compared to decision tree but reduces the variance dramatically.

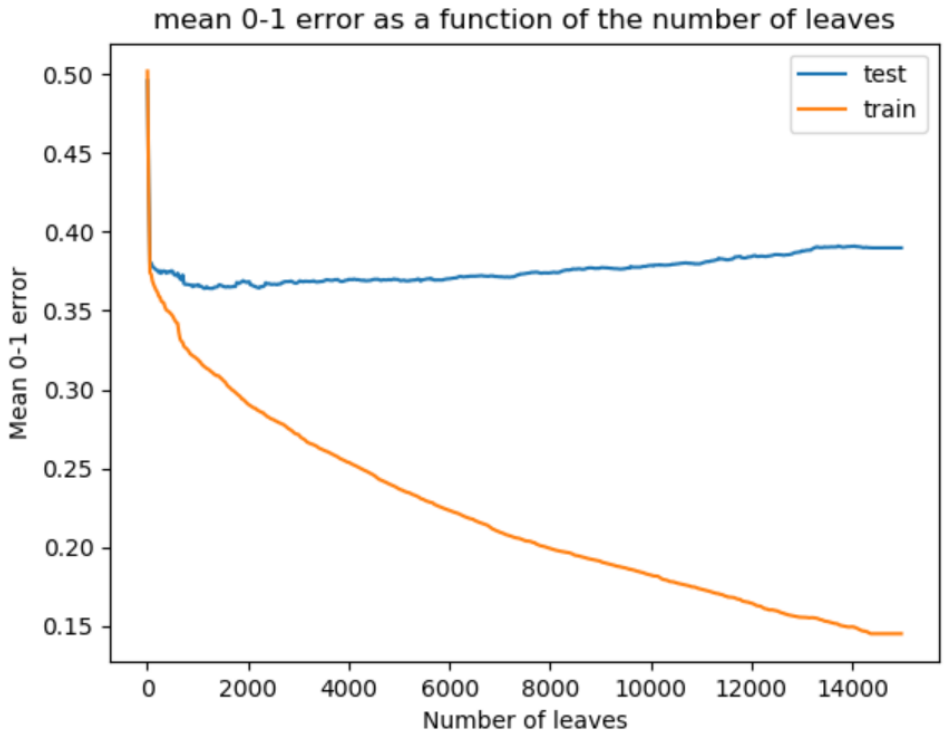
I have looked for the best parametrization for forest, in doing so I'm risking overfitting on the testing data and that's the reason why we should use a vault. In
order to do that I firstly focused on a random forest containing only
one tree (= a decision tree).

Here is a graph of mean 0-1 error as a function of the number of the leaves.
Max_feature = 'sqrt' (=The number of features to consider when looking for the best split)



Idan Yamin

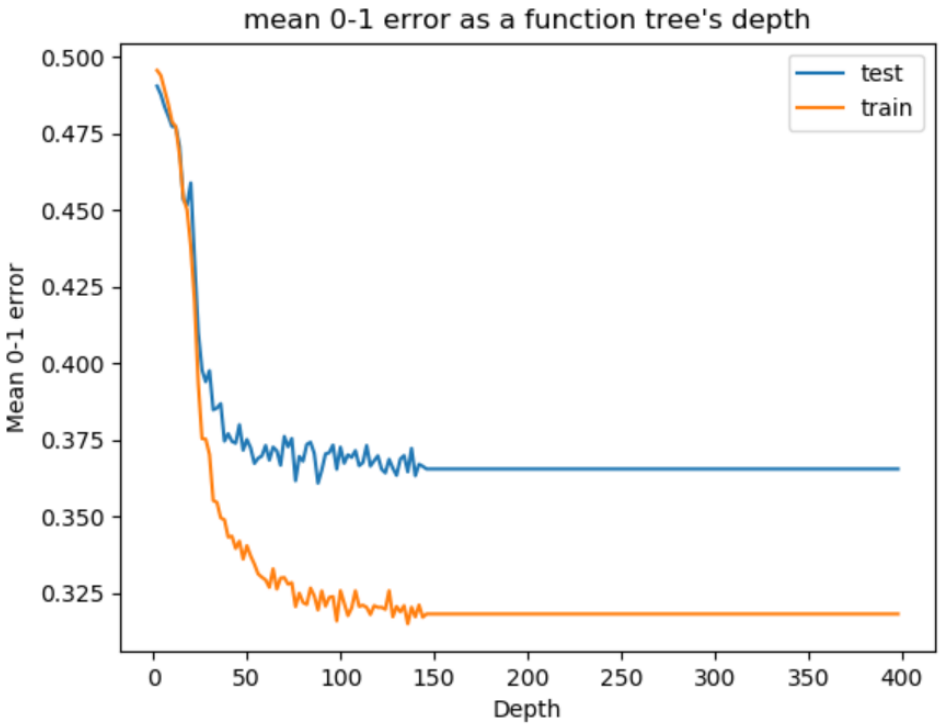
Same graph but Max_feature = 'log2'



As we can see when considering 'sqrt' features we get lower test error, but the disadvantage of using max_feature='sqrt' is that it is computationally harder I
choose to use 'log2' instead.
We can also conclude that around 1000 leaves we get the optimal solution,
We don't overfit the data in one hand and on the other hand we get the lowest testing error.

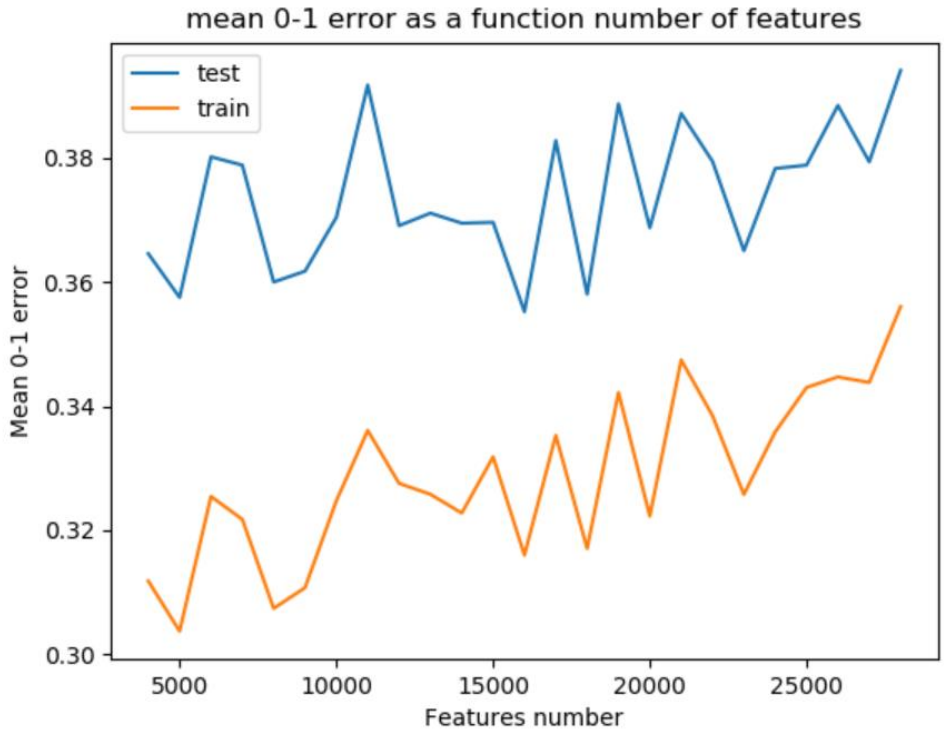
Idan Yamin

Mean 0-1 error as a function of tree's depth.



Around 170 the mean error converges.

Mean 0-1 error as a function of the number of features.

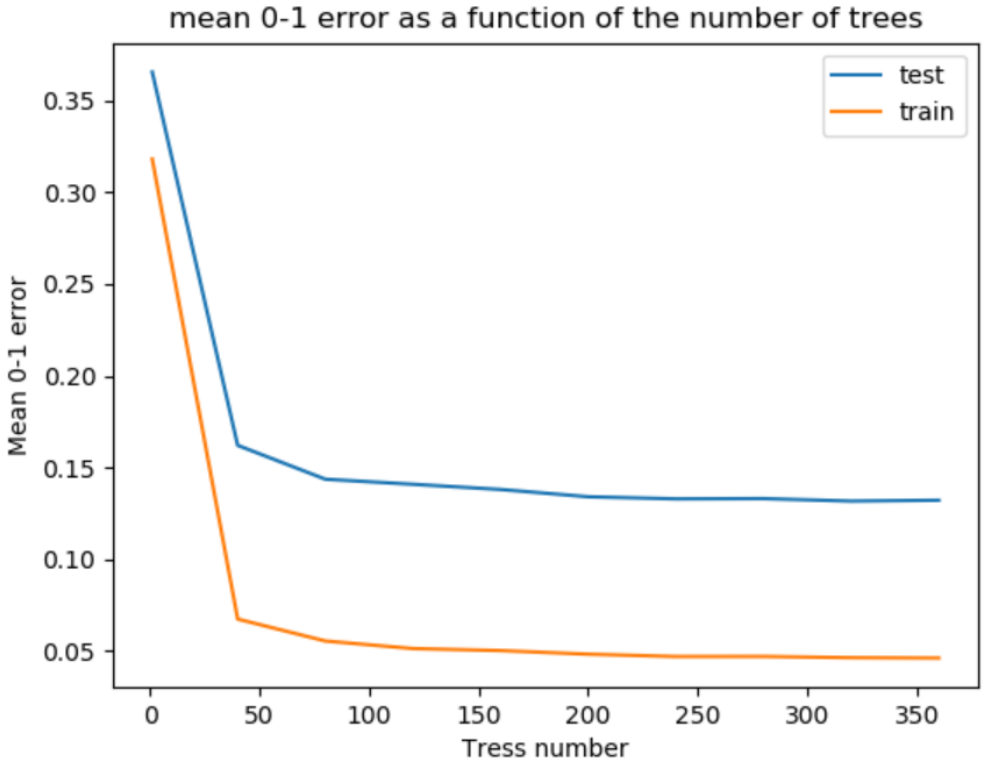


From the graph above there is no clear answer on the number of features we should use.

Idan Yamin

1,600 features seem like a good guess.

Now that we found parameters that hopefully will minimize the test error for a single tree.
Let's try to add more trees and see how the test error will decrease.



As we can see the test error starts to slowly converge when using
120 trees.

The test error is worse than the baseline classifier. Meaning it is not good enough.
Let's try to boost this classifier using Adaboost with 50 iterations.

The score of a prediction rule is defined as 1 minus the zero-one loss.
The score of the boosted random forest is: 0.9041077441077441

Because of the process I used here for finding an optimal solution
there is a risk of overfitting over the testing data.
In order to check that I didn't overfit I opened the vault and trained the classifier over all the data that is not in the vault.

The score on the vault is: 0.9063745019920319 it's better than the
score on the testing data meaning it's more likely that the classifier didn’t overfit.

Idan Yamin

User's guide

Notice I have added an option to create your own testing data and run it on the random forest classifier.

All the results and graphs can be generated by a python program I wrote.
In order to run it you should have the files: IMDB_Dataset.csv, graphs_and_results.py, menu.py and main.py
You should run the file main.py
When running it you should see this menu:

```
1.Base line classifier (Logistic Regression)
2.Linear classifier mean 0-1 loss (an example of bad classifier)
3.Ridge regression
4.Lasso regression, plot regularization path
5.Adaboost
6.Random Forest
```

You should insert the number of the thing that you want to run.
For example, if I want to plot ridge regression regularization path that I have shown.
I need to insert 3→enter
Then this menu will pop up

```
You choose Ridge Regression
1.plot graph of 0-1 mean loss as a function of lambda
2.plot regularization path
```

Then you should insert 2→enter and wait for the graph.

The finial classifier I showed was the boosted random forest.
In order to run it you should have the files: DataVault.csv, adabost_50_final.pickle, bag_of_words_opt.pickle and random_forest_solustion.py

You can run the program with one argument that states the path
of your testing file.

Your testing file should be a .csv file. It should have one column of review and one column of sentiment. Each row contains one review and one sentiment (positive/negative).
I have included a file for example my_reviews.csv

When running it you should see this menu:

```
1.Get score on vault
2.Get score on your reviews
```

Notice if you choose 2 you must have a valid path to a csv file in a format like the file my_reviews.csv