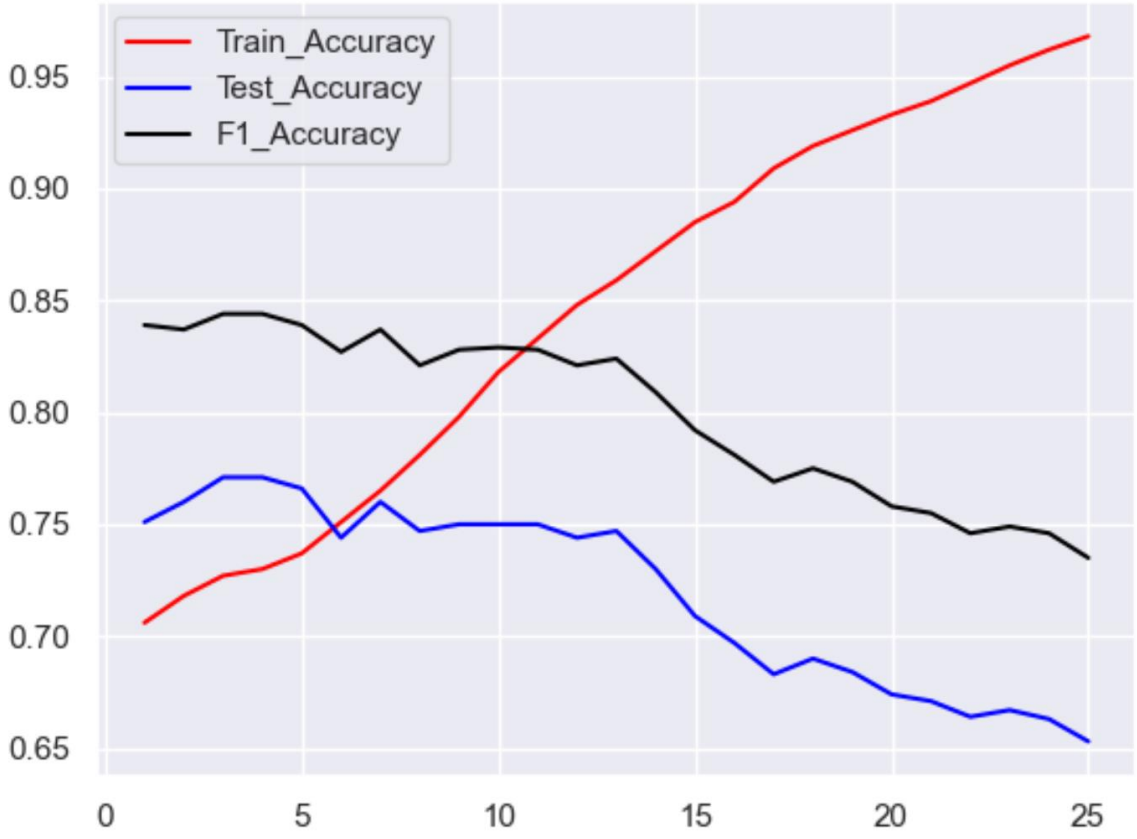
Assignment3 Report

Part1

(d) Now, add a function in main for creating trees with depths ranging from 1 to 25 (inclusive). Plot and explain the behavior of train/testing performance against the depth. That is, plot accuracy versus number of trees, as well as F1 score vs number of trees. (F1 score is a weighted average of precision and recall, I believe you've covered precision/recall already).



Base on the graph, the more depth we have, the more training accuracy we get, it means that we have an overfitting problem. Our model may fit the training set well, but it will be bad for the testing set.

The trend of testing accuracy and F1 accuracy are similar, the more depth it has, the less accurate it is. It makes sense that the trend are similar since F1 and testing accuracy are just different measurement for the same testing set. If the depth is lower, it will not have overfitting problem and it gives higher accuracy for F1.

(e) Report the depth that gives the best validation accuracy? You will probably hit a point where it will not need to be deeper, and it's best to use the simplest version.

```
The best Depth is  3 with F1 accuracy  0.844
```

f) What is the most important feature for making a prediction? How can we find it? Report the name of it (see data dictionary) as well as the value it takes for the split.

```
Feature: 8 - Description: White alone, percent, 2014
```

The most information gain should be the most important feature. It should be the first feature to split data into 2 groups, which is the feature for the root of tree.

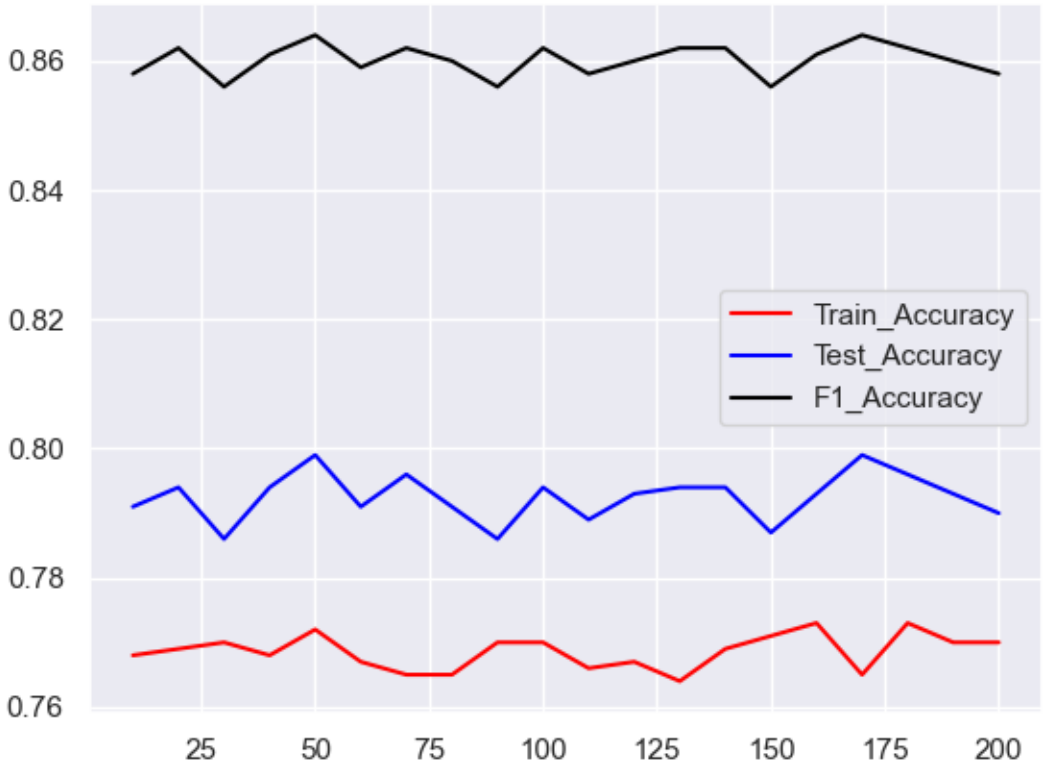
Par2

(b) For $max\_depth = 7$, $max\_features = 11$ and $n\_trees \in [10, 20, ..., 200]$, plot the train and testing accuracy of the forest versus the number of trees in the forest n. Please also plot the train and testing F1 scores versus the number of trees.

(c) What effect does adding more trees into a forest have on the train/testing performance? Why?

	N_Trees	Train_Accuracy	Test_Accuracy	F1_Accuracy
0	10	0.768	0.791	0.858
1	20	0.769	0.794	0.862
2	30	0.770	0.786	0.856
3	40	0.768	0.794	0.861
4	50	0.772	0.799	0.864
5	60	0.767	0.791	0.859
6	70	0.765	0.796	0.862
7	80	0.765	0.791	0.860
8	90	0.770	0.786	0.856
9	100	0.770	0.794	0.862
10	110	0.766	0.789	0.858
11	120	0.767	0.793	0.860
12	130	0.764	0.794	0.862
13	140	0.769	0.794	0.862
14	150	0.771	0.787	0.856
15	160	0.773	0.793	0.861
16	170	0.765	0.799	0.864
17	180	0.773	0.796	0.862
18	190	0.770	0.793	0.860
19	200	0.770	0.790	0.858

The best N_Trees is 50 with F1 accuracy 0.864



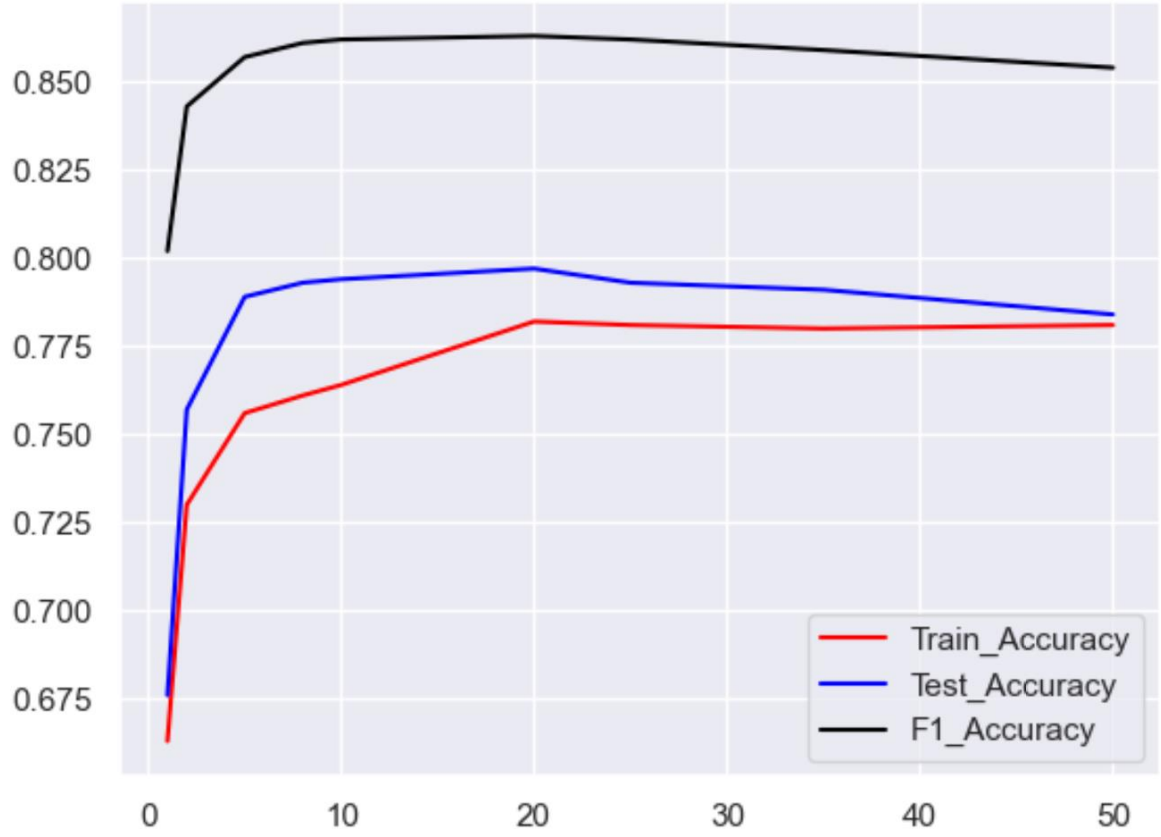
Base on the graph, the accuracy may go up or down by the number of trees increase. The more trees we have, the bigger the voting group it is, which means every decision tree's vote weight less. The more trees we have, the correlation between trees will be higher. The less tress we have, the correlation between tress will be lower. We want to have high variances in random forest and then average the variances will help our model to be closer to the target.

As a result, our best 'N_trees' is 50 and its F1 accuracy is 86.4%. It makes sense that our number of trees should not be too big in case the correlation of trees is too high, the number of trees should not be too low in case we don't have enough variances from the dataset.

(d) Repeat above experiments for $max\_depth = 7$, $n = 50$ trees, and $max\_features \in [1, 2, 5, 8, 10, 20, 25, 35, 50]$. How does $max\_features$ change the train/validation accuracy? Why?

	MAX_Features	Train_Accuracy	Test_Accuracy	F1_Accuracy
0	1	0.675	0.691	0.810
1	2	0.730	0.759	0.844
2	5	0.751	0.786	0.857
3	8	0.764	0.796	0.862
4	10	0.759	0.793	0.861
5	20	0.781	0.790	0.858
6	25	0.776	0.787	0.858
7	35	0.785	0.794	0.861
8	50	0.785	0.789	0.858

The best MAX_Features is 8 with F1 accuracy 0.862



If max_features is too high, we will not make good random decision trees with good variances, every tree will be similar. If max_features is too low, we will not find the good feature to split the dataset, and the accuracy will be bad.

In this graph, by max_features increase, the accuracy will increase and then be stable and slowly decrease.

Our best MAX_Features is 8 and its accuracy is 86.2%

(e) Optional: try to find the best combination of the three parameters using a similar idea.

(f) With your best result, run 10 trials with different random seeds (for the data/feature sampling you can use the same seed) and report the individual train/testing accuracy's and F1 scores, as well as the average train/testing accuracy and F1 scores across the 10 trials. Overall, how do you think randomness affects the performance?

```
pool = mp.Pool(mp.cpu_count())

Ntree_Task=[x for x in range(150,300,10)]
Max_features_Task=[5,6,7,8,9,10,11,12]
max_depth_Task=[10,12,14,16,18,20]

all_tasks=[(x_train, y_train, x_test, y_test,x,y,z) for x in Max_features_Task for y in Ntree_Task for z in max_depth_Task]

result=pool.starmap(rf_tune_all,all_tasks)
print(all_tasks)
print(result)

bestAccuracy=max(result)
bestcombo=all_tasks[result.index(bestAccuracy)]

print("The best combo of (Features,Ntree,Depth) is:")
print(bestcombo[-3],bestcombo[-2],bestcombo[-1])
print(bestAccuracy)
pool.close()
```

```
The best combo of (Features,Ntree,Depth) is:
8 210 15
0.865
Done
```

I used parallel programming to run a bunch of combos, the best accuracy I can get is 86.5% with Max_features=8, N_tree=210, Depth=15

Seed	F1	Train	Test
1	0.8580	0.8710	0.7910
11	0.8570	0.8690	0.7900
21	0.8620	0.8710	0.7970
31	0.8580	0.8700	0.7930
41	0.8590	0.8670	0.7930
51	0.8620	0.8670	0.7970
61	0.8570	0.8680	0.7900
71	0.8570	0.8680	0.7900
81	0.8550	0.8690	0.7870
91	0.8600	0.8710	0.7940
Average	0.8585	0.8691	0.7922

For running 10 different seeds, I don't think random seeds will impact the result much. Random forest will be random, however, if you use the same features, it will not be too different.

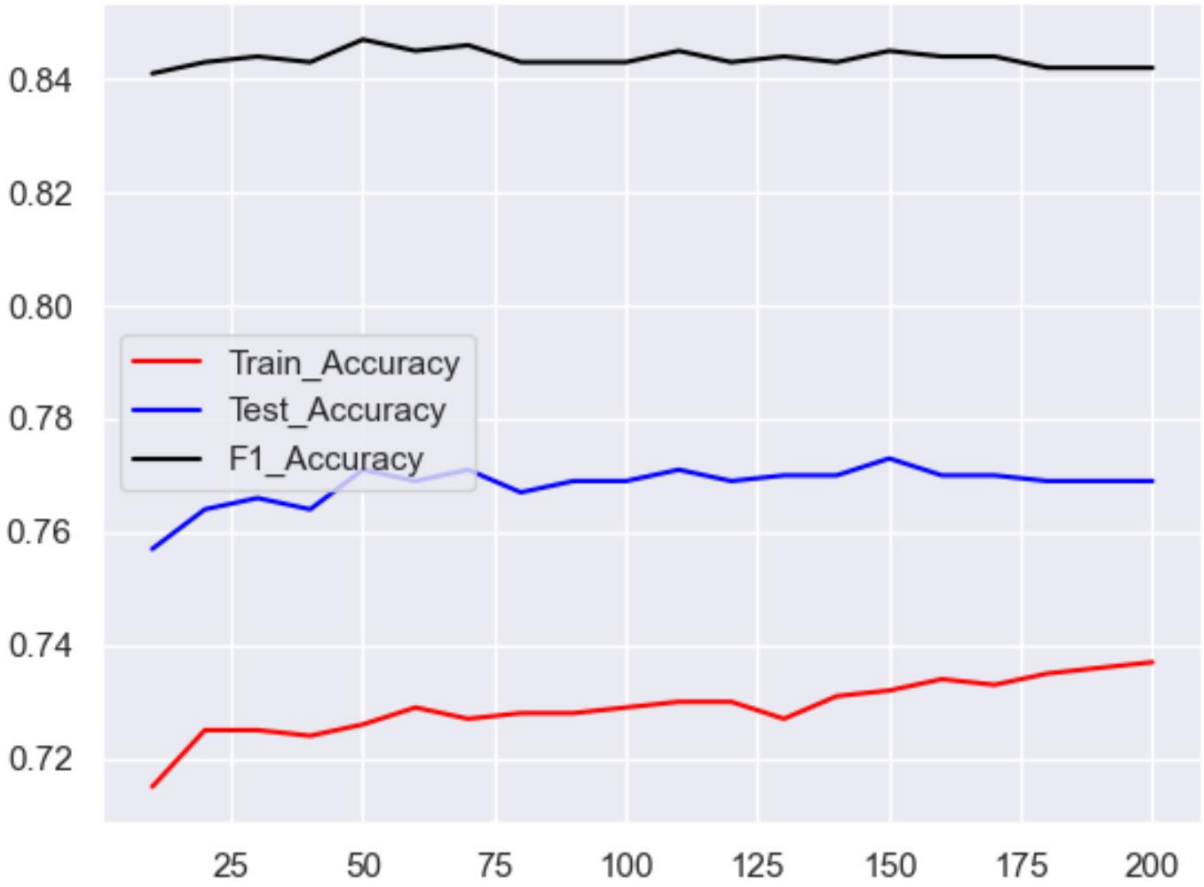
Part3

(e) Report the train and validation accuracy for $L \in [10, 20, ..., 200]$.

(f) Plot the train/test accuracy and train/test F1 scores of AdaBoost against the parameter L.

	L	Train_Accuracy	Test_Accuracy	F1_Accuracy
0	10	0.715	0.757	0.841
1	20	0.725	0.764	0.843
2	30	0.725	0.766	0.844
3	40	0.724	0.764	0.843
4	50	0.726	0.771	0.847
5	60	0.729	0.769	0.845
6	70	0.727	0.771	0.846
7	80	0.728	0.767	0.843
8	90	0.728	0.769	0.843
9	100	0.729	0.769	0.843
10	110	0.730	0.771	0.845
11	120	0.730	0.769	0.843
12	130	0.727	0.770	0.844
13	140	0.731	0.770	0.843
14	150	0.732	0.773	0.845
15	160	0.734	0.770	0.844
16	170	0.733	0.770	0.844
17	180	0.735	0.769	0.842
18	190	0.736	0.769	0.842
19	200	0.737	0.769	0.842

The best L is 50 with the best accuracy: 0.847



When L=50, we get the best accuracy 84.7%

(b) **Write a summary of ideas you have for handling the imbalanced classes in our data.** There's no "right answer" here, but you should think about things we could use. I've added F1 score as a metric, which is coarse, but better than just accuracy. What else could I have done to see how good the classifier is on the testing data? I think this is important because sometimes (very, very often) in the real world we see much worse class imbalance than this, and we don't want to blindly report metrics like accuracy if we're reporting the wrong ones. Feel free to report ideas from the article, and if you're not familiar with precision/recall, refresh on it via Wikipedia or slides.

In order to solve the problem of unbalanced dataset, we can under sample the majority classes base on 'Bayesian argument of Wallace et al'. Under sample will generate more variances, but we can ensemble all classifier to average the variances.

'Tomek links' is another good idea to remove the noisy pairs of the data and it will make the border clear so that it is easier for ML algorithm to make the decision boundary.

'SMOTE and descendants' is a way of over sample, it will generate more sample for the minority group.

‘Adjusting class weights’ will make minority samples more important, as a result, the decision boundary will bias to majority samples less and make a more fair judgement.

I feel for this project, ‘SMOTE and descendants’ is a good idea, since people who vote for the same party will have close similarities.