COGS 118A Final Project

Fei Yin A15555426

Abstract

Numerous supervised machine learning algorithms have been developed, implemented, and refined throughout the last several decades. This project applies a small subset of the well known supervised machine learning methods on several datasets produced from entirely different fields. The purpose of this project is to reproduce and compare to the results of similar works done in the past.

1 Introduction

This project evaluates the binary classification performance of 6 different supervised machine learning algorithms: logistic regression, perceptron, SVM, decision tree, random forest, and gradient boosting classifier. Variations among each of the algorithms are explored. The datasets used are: Adult (ADULT) [1], Occupancy Detection (OCCDET) [2], HTRU2 [3], and Activity recognition with healthy older people using a batteryless wearable sensor (ACTREC) [4]. The performance of the algorithms on the dataset are measured based on three metrics: accuracy, F1 score, and ROC area.

2 Methodology

2.1 Learning Algorithms

The scikit-learn implementations of the following algorithms are explored.

Logistic Regression (LOGREG)[5]: We use the Newton-CG solver with L2 penalty and range the C parameter from 10^{-3} to 10^{3} by factors of 10.

Perceptron (PERCEP)[6]: We use both L1 and L2 penalties and range the alpha parameter from 10^{-3} to 10^{3} by factors of 10 for each of the two penalties.

SVMs[7]: We use both linear (LINSVM) and RBF (RBFSVM) kernels. For both kernels, we range the C parameter from 10^{-3} to 10^{3} by factors of 10. For the RBF kernel, we also range the gamma parameter from 10^{-3} to 10^{1} by factors of 10.

Decision Tree (DECTRE)[8]: We use both the gini and entropy criterions. For both criterions, we vary the maximum depth {10, 50, 100, 500, 1000} and cost complexity pruning {0, 1e-1, 1e0, 1e1}.

Random Forest (RANFOR)[9]: We use both the gini and entropy criterions. For both criterions, we vary the number of estimators {10, 50, 100, 500, 1000} and cost complexity pruning {0, 1e-1, 1e0, 1e1}.

Gradient Boosting Classifier (GRABOO)[10]: We use the exponential loss and range the learning rate from 10^{-3} to 10^{1} by factors of 10 and vary the number of estimators {10, 50, 100, 500, 1000}.

2.2 Performance Metrics

As mentioned above, the performance metrics used are: accuracy, F1 score, and ROC area. Among the three metrics, accuracy gives the most straightforward measure of the proportion of correct classifications whereas F1 score and ROC area give us more insight about the trade off between obtaining more correct classifications and allowing more false positives or negatives.

Preprint. Under review.

2.3 Datasets

We evaluate the algorithms on the four different datasets shown in Table 1. For all four datasets, we one-hot encode the categorical attributes. The second column in Table 1 shows the number of attributes after encoding. For ADULT, which has an excessive number of attributes, we apply principal component analysis to the dataset and map it to the subspace spanned by the top 30 principal components, which are verified to represent approximately 90% of total variance ratio. For ACTREC, which has more than one categories of activity as the result of classification, we use the last category, 'ambulating', as the label for our binary classification task on this dataset.

Table 1: Description of Datasets

Dataset	#ATTR	Train Size	Test Size	%POZ
ADULT	30/105	5000	27560	24%
OCCDET	5	5000	15560	23%
HTRU2	8	5000	12897	9%
ACTREC	10	5000	47482	4%

3 Experiment

To test every algorithm's performance on every dataset, we first split each of the four datasets into five versions of training and testing sets. For each dataset, we randomly sample 5000 points with replacement as a training set, and let the rest of the data be a testing set. This random sampling process is done five times, each with a different seed. Since we have four datasets in the beginning, we now have 20 pairs of training and testing sets. Next, every algorithm is run on every pair of training and testing sets under the three different metrics. For each algorithm-set-metric combination, a 5-fold cross validation is performed to search for the best hyperparameters from the options detailed earlier.

To visualize the hyperparameters' influence on performance, heat maps for each algorithm's hyperparameter search process is recorded as Figure 1 to 27 in the Appendix section. From the heat maps, we see the following patterns:

- For logistic regression, higher C gives better results, but the improvements are significantly decreasing as we reach the highest end of our search range. This suggests that C = 1000 is approximately the best hyperparameter in this case.
- For perceptron, L1 penalty with small alpha gives better results. This suggests that we would benefit from further decreasing the search range of alpha.
- For linear SVM, higher C gives better results, but the improvements are significantly decreasing as we reach the highest end of our search range. This suggests that C = 1000 is approximately the best hyperparameter in this case.
- For RBF SVM, higher C generally gives better results, with the best performance happening at C = 100 or 1000. On the other hand, with respect to gamma, the best performance seems to greatly depend on the metric being used.
- For decision tree, performance under gini and entropy criterions are almost the same. However, adding cost complexity pruning degrades performance tremendously. This is also the case with random forest. With respect to maximum depth, smaller value yields better performance, although the difference is minor.
- For random forest, performance under gini and entropy criterions are almost the same with no cost complexity pruning. However, with cost complexity pruning set to 0.1, the entropy criterion performs better, or, to be more specific, less bad. With respect to the number of estimators, larger value yields better performance, although the difference is minor.
- For gradient boosting classifier, the best performance happens around 500 estimators and learning rate = 0.1.

With the best hyperparameters in the given search range, performance on the training and testing data is evaluated and recorded. All performance scores obtained for the testing data are shown in Table 5 and 6 in the Appendix section. By averaging over the datasets and trials, we obtain a 2D array that

shows each algorithm's average testing performance under the three metrics. This is shown as Table 2. On the other hand, by averaging over the metrics, we obtain a 2D array that shows each algorithm's average testing performance under the four datasets. This is shown as Table 3. The last column of Table 2 and 3 show the average testing performance of each algorithm over both the metrics and the datasets.

Furthermore, in Table 2 and 3, the best performing algorithm under each metric or dataset is **boldfaced**. To see if other algorithms' performances are distinguishable from the best performing algorithm in each column, pair-wise t-tests are performed between the best performing algorithm and all the others for each column in both tables. The algorithms with statistically indistinguishable performance from the best performing algorithm, that is, if their corresponding pair-wise t-tests output $p > 0.05$, we mark those algorithms with * in the corresponding column.

Table 2: Testing scores for each algorithm by metric (average over four datasets)

ALGORITHM	ACC	F1	ROC	MEAN
LOGREG	0.943	0.757	0.834	0.845
PERCEP	0.906	0.576	0.757	0.746
LINSVM	0.943	0.767	0.838	0.849
RBFSVM	0.944	0.781	0.855	0.860
DECTRE	0.936	0.773	0.864*	0.858
RANFOR	0.942	0.799*	0.860	0.867
GRABOO	**0.946**	**0.802**	**0.868**	**0.872**

Table 3: Testing scores for each algorithm by dataset (average over three metrics)

ALGORITHM	ADULT	OCCDET	HTRU2	ACTREC	MEAN
LOGREG	**0.719**	0.985	0.926*	0.750	0.845
PERCEP	0.649	0.928*	0.857*	0.552	0.746
LINSVM	0.711*	0.985	0.925*	0.777	0.849
RBFSVM	0.716*	0.986*	0.925*	0.813	0.860
DECTRE	0.693	0.985	0.920	0.832	0.858
RANFOR	0.700	**0.988**	0.927*	0.853*	0.867
GRABOO	0.714*	0.987	**0.928**	**0.859**	**0.872**

4 Discussion

From the results detailed in Table 2 and 3, we can easily see that the performance under different metrics and datasets are very different. Specifically, performance of all the algorithms evaluated with accuracy is the highest among the three metrics, and performance of all the algorithms on the OCCDET dataset is the highest. On the other hand, performance of all the algorithms on the ADULT dataset is the lowest. It seems that datasets with fewer attributes perform better. This may very likely be caused by the curse of dimensionality. Also, it is worth noting that all datasets used are more or less biased toward the negative class, as shown in Table 1, but this does not seem to be making as big of an impact as the number of attributes.

Furthermore, from Table 2 and 3, we see that gradient boosing classifier is performing the best most of the times, although with a minor $< 1\%$ lead, on average, over random forest, the next best performing algorithm. However, perceptron's performance is significantly lower than all other algorithms, about 10% lower than the second worst. This does not seem to be aligned with the No Free Lunch Theorem, except for the case where logistic regression achieves the highest performance on the ADULT dataset.

5 Conclusion

In this project, we implemented 6 algorithms on 4 datasets under 3 metrics. Although we selected datasets produced by vastly different fields, compared to the amount of data produced, what we have surveyed is a minute subset. However, when compared to previous work [11], our results mostly line up that boosting algorithm generally performs the best, followed by random forest.

6 Bonus Points

- Principal component analysis, which is not included in the course curriculum, is performed on one of the datasets. By reducing the dimensionality of the attribute space, computation time is greatly reduced, and performance on that dataset may have been increased by alleviating the curse of dimensionality. However, given the results shown in the tables, reducing to 30 dimensions to achieve 90% variance ratio may have been too greedy of a decision.
- Heat maps for all algorithms under all metrics are drawn to better show the hyperparameter searching process. This is a large amount of work, given that the dimensions, parameters, and search ranges vary from one algorithm to another.

References

[1] URL: https://archive.ics.uci.edu/ml/datasets/Adult.

[2] URL: https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+.

[3] URL: https://archive.ics.uci.edu/ml/datasets/HTRU2.

[4] URL: https://archive.ics.uci.edu/ml/datasets/Activity+recognition+with+healthy+older+people+using+a+batteryless+wearable+sensor.

[5] URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html?highlight=logistic#sklearn.linear_model.LogisticRegression.

[6] URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html?highlight=perceptron#sklearn.linear_model.Perceptron.

[7] URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html?highlight=svc#sklearn.svm.SVC.

[8] URL: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html?highlight=decision%20tree#sklearn.tree.DecisionTreeClassifier.

[9] URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html?highlight=randomforest#sklearn.ensemble.RandomForestClassifier.

[10] URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html?highlight=gradientboost#sklearn.ensemble.GradientBoostingClassifier.

[11] "Alexandru Niculescu-Mizil" "Rich Caruana". “An Empirical Comparison of Supervised Learning Algorithms”. In: (2006).

7 Appendix

Table 4: Training scores for each algorithm by metric (average over four datasets)

ALGORITHM	ACC	F1	ROC	MEAN
LOGREG	0.943	0.753	0.831	0.843
PERCEP	0.918	0.623	0.780	0.773
LINSVM	0.943	0.764	0.836	0.848
RBFSVM	0.944	0.777	0.851	0.857
DECTRE	0.935	0.779	0.862	0.859
RANFOR	0.941	0.795	0.858	0.865
GRABOO	0.945	0.799	0.866	0.870

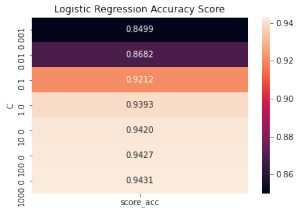


Figure 1: LOGREG ACC

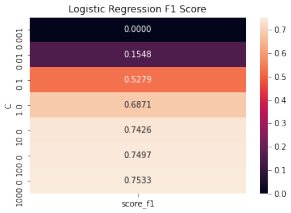


Figure 2: LOGREG F1

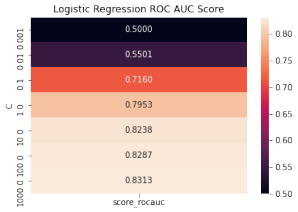


Figure 3: LOGREG ROC

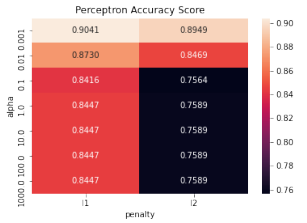


Figure 4: PERCEP ACC

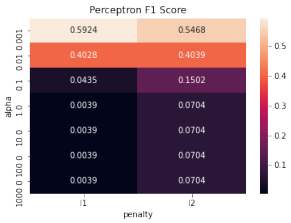


Figure 5: PERCEP F1

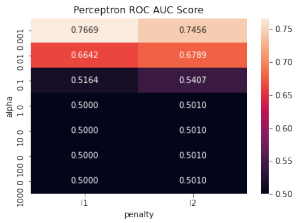


Figure 6: PERCEP ROC

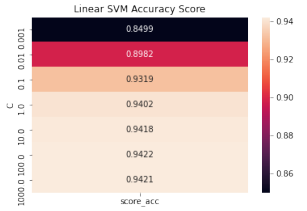


Figure 7: LINSVM ACC

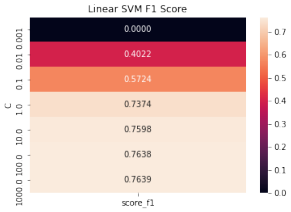


Figure 8: LINSVM F1

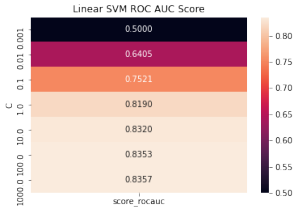


Figure 9: LINSVM ROC

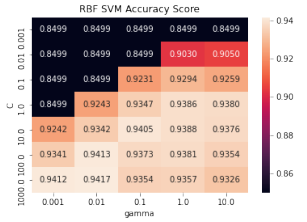


Figure 10: RBFSVM ACC

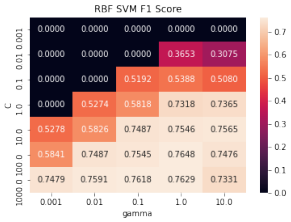


Figure 11: RBFSVM F1

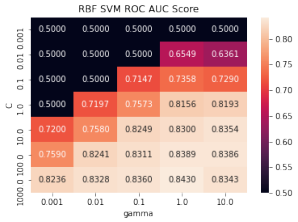


Figure 12: RBFSVM ROC

Table 5: Raw test set performance for ADULT and OCCDET datasets

ALGORITHM	ADULT			OCCDET		
	ACC	F1	ROC	ACC	F1	ROC
Trial 1						
LOGREG	0.829	0.598	0.726	0.988	0.976	0.991
PERCEP	0.649	0.602	0.745	0.985	0.968	0.989
LINSVM	0.828	0.595	0.724	0.988	0.975	0.991
RBFSVM	0.828	0.601	0.729	0.990	0.978	0.993
DECTRE	0.797	0.564	0.708	0.989	0.976	0.991
RANFOR	0.812	0.568	0.711	0.991	0.981	0.991
GRABOO	0.828	0.601	0.730	0.990	0.979	0.992
Trial 2						
LOGREG	0.828	0.602	0.730	0.989	0.977	0.992
PERCEP	0.787	0.531	0.693	0.988	0.975	0.988
LINSVM	0.829	0.612	0.740	0.989	0.976	0.991
RBFSVM	0.828	0.610	0.736	0.990	0.979	0.992
DECTRE	0.803	0.578	0.724	0.989	0.977	0.991
RANFOR	0.809	0.577	0.718	0.990	0.980	0.990
GRABOO	0.828	0.594	0.723	0.989	0.977	0.987
Trial 3						
LOGREG	0.828	0.602	0.729	0.988	0.974	0.990
PERCEP	0.771	0.412	0.620	0.913	0.777	0.822
LINSVM	0.826	0.571	0.711	0.988	0.975	0.990
RBFSVM	0.826	0.599	0.727	0.988	0.974	0.992
DECTRE	0.799	0.564	0.700	0.988	0.975	0.990
RANFOR	0.813	0.580	0.715	0.991	0.981	0.990
GRABOO	0.829	0.574	0.722	0.990	0.979	0.990
Trial 4						
LOGREG	0.829	0.605	0.731	0.989	0.976	0.991
PERCEP	0.766	0.585	0.735	0.902	0.825	0.936
LINSVM	0.822	0.587	0.721	0.988	0.975	0.991
RBFSVM	0.826	0.600	0.729	0.988	0.976	0.991
DECTRE	0.799	0.547	0.716	0.989	0.975	0.991
RANFOR	0.814	0.573	0.712	0.992	0.981	0.991
GRABOO	0.830	0.597	0.729	0.991	0.981	0.991
Trial 5						
LOGREG	0.826	0.591	0.722	0.990	0.978	0.992
PERCEP	0.792	0.414	0.627	0.978	0.894	0.985
LINSVM	0.826	0.564	0.705	0.989	0.976	0.991
RBFSVM	0.827	0.566	0.707	0.991	0.980	0.993
DECTRE	0.796	0.579	0.724	0.990	0.978	0.991
RANFOR	0.812	0.572	0.714	0.992	0.982	0.991
GRABOO	0.826	0.579	0.717	0.991	0.981	0.991

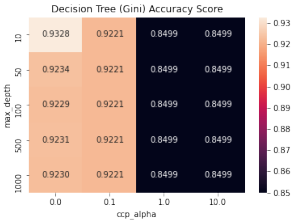


Figure 13: DECTRE gini ACC

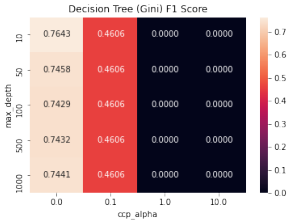


Figure 14: DECTRE gini F1

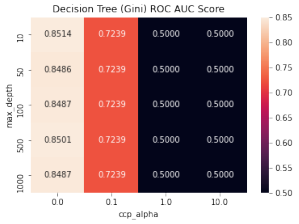


Figure 15: DECTRE gini ROC

Table 6: Raw test set performance for HTRU2 and ACTREC datasets

ALGORITHM	HTRU2			ACTREC		
	ACC	F1	ROC	ACC	F1	ROC
Trial 1						
LOGREG	0.980	0.882	0.912	0.976	0.554	0.698
PERCEP	0.976	0.855	0.891	0.885	0.025	0.506
LINSVM	0.979	0.876	0.907	0.979	0.618	0.726
RBFSVM	0.980	0.886	0.915	0.980	0.667	0.784
DECTRE	0.978	0.874	0.910	0.979	0.666	0.826
RANFOR	0.980	0.886	0.922	0.983	0.731	0.801
GRABOO	0.980	0.885	0.923	0.983	0.744	0.824
Trial 2						
LOGREG	0.980	0.885	0.916	0.977	0.571	0.706
PERCEP	0.977	0.868	0.762	0.963	0.058	0.515
LINSVM	0.980	0.888	0.918	0.979	0.624	0.729
RBFSVM	0.979	0.877	0.910	0.979	0.683	0.790
DECTRE	0.977	0.870	0.906	0.978	0.675	0.852
RANFOR	0.980	0.882	0.918	0.984	0.771	0.829
GRABOO	0.980	0.886	0.923	0.985	0.787	0.854
Trial 3						
LOGREG	0.980	0.882	0.914	0.978	0.582	0.710
PERCEP	0.975	0.852	0.856	0.963	0.353	0.690
LINSVM	0.980	0.883	0.915	0.980	0.628	0.732
RBFSVM	0.979	0.875	0.906	0.980	0.662	0.782
DECTRE	0.977	0.869	0.911	0.981	0.706	0.833
RANFOR	0.979	0.880	0.910	0.986	0.771	0.822
GRABOO	0.980	0.883	0.917	0.984	0.762	0.847
Trial 4						
LOGREG	0.980	0.883	0.912	0.977	0.572	0.706
PERCEP	0.947	0.598	0.714	0.970	0.130	0.736
LINSVM	0.980	0.880	0.908	0.979	0.623	0.728
RBFSVM	0.980	0.891	0.917	0.979	0.677	0.801
DECTRE	0.978	0.873	0.916	0.981	0.705	0.848
RANFOR	0.980	0.886	0.917	0.986	0.774	0.810
GRABOO	0.979	0.879	0.912	0.985	0.763	0.841
Trial 5						
LOGREG	0.980	0.883	0.914	0.977	0.563	0.699
PERCEP	0.967	0.783	0.826	0.967	0.018	0.500
LINSVM	0.980	0.882	0.912	0.979	0.621	0.727
RBFSVM	0.980	0.882	0.916	0.978	0.657	0.801
DECTRE	0.978	0.873	0.916	0.977	0.637	0.834
RANFOR	0.979	0.883	0.920	0.983	0.747	0.821
GRABOO	0.980	0.886	0.926	0.982	0.722	0.826

Table 7: Pair-wise T Test for Table 2: testing scores for each algorithm by metric (average over four datasets)

ALGORITHM	ACC	F1	ROC
LOGREG	0.000	0.000	0.000
PERCEP	0.018	0.000	0.000
LINSVM	0.010	0.000	0.000
RBFSVM	0.008	0.000	0.001
DECTRE	0.000	0.000	0.126
RANFOR	0.000	0.455	0.004
GRABOO	nan	nan	nan

Table 8: Pair-wise T Test for Table 3: testing scores for each algorithm by dataset (average over three metrics)

ALGORITHM	ADULT	OCCDET	HTRU2	ACTREC
LOGREG	nan	0.014	0.134	0.000
PERCEP	0.016	0.108	0.053	0.001
LINSVM	0.157	0.003	0.131	0.000
RBFSVM	0.449	0.139	0.293	0.001
DECTRE	0.003	0.010	0.015	0.001
RANFOR	0.000	nan	0.522	0.184
GRABOO	0.106	0.032	nan	nan

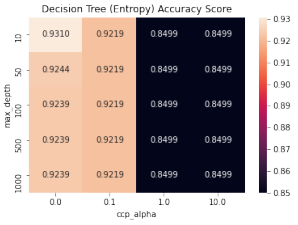


Figure 16: DECTRE entropy ACC

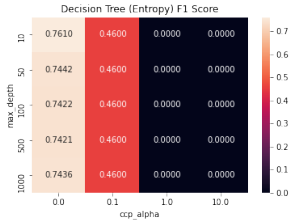


Figure 17: DECTRE entropy F1

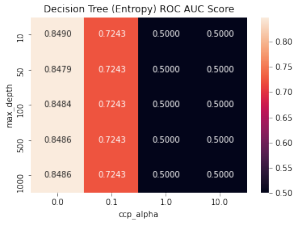


Figure 18: DECTRE entropy ROC

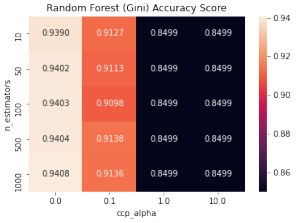


Figure 19: RANFOR gini ACC

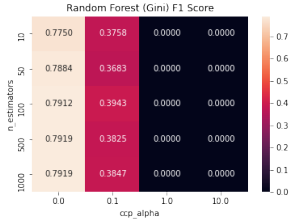


Figure 20: RANFOR gini F1

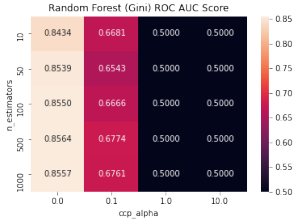


Figure 21: RANFOR gini ROC

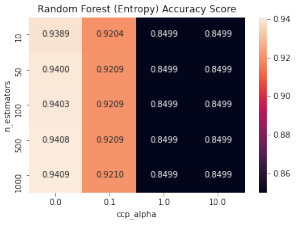


Figure 22: RANFOR entropy ACC

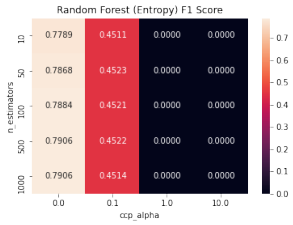


Figure 23: RANFOR entropy F1

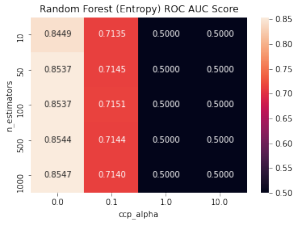


Figure 24: RANFOR entropy ROC

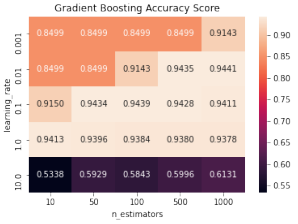


Figure 25: GRABOO ACC

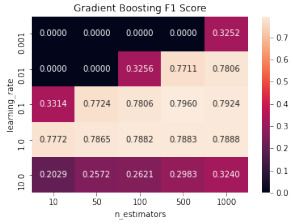


Figure 26: GRABOO F1

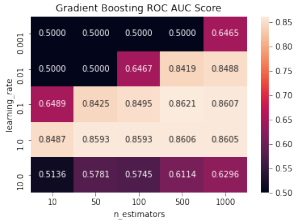


Figure 27: GRABOO ROC

data_cleaning

March 17, 2021

[1]:
```
import numpy as np
import pandas as pd
from sklearn import preprocessing
import glob
```

0.0.1 Clean Adult Dataset

https://archive.ics.uci.edu/ml/datasets/Adult

[2]:
```
## Read dataset
df_adult = pd.read_csv('datasets/original/adult_train.csv')
```

[3]:
```
df_adult.columns = ['age', 'workclass', 'fnlwgt', 'education', 'education_num',␣
↪'marital-status', 'occupation',
                    'relationship', 'race', 'sex', 'capital-gain',␣
↪'capital-loss', 'hours-per-week', 'native-country',
                    'salary']

#df_adult.replace({'?': np.nan}).dropna()
```

[4]:
```
## Apply one hot encoding to categorical attributes
categorical_attributes = ['workclass', 'education', 'marital-status',␣
↪'occupation', 'relationship', 'race', 'sex',
                          'native-country', 'salary']

for categorical_attribute in categorical_attributes:

    ## Strip white space while encoding
    one_hot = pd.get_dummies(df_adult[categorical_attribute].str.strip(),␣
↪prefix=categorical_attribute)

    ## Drop columns that are encoded for unknown values
    try:
        one_hot = one_hot.drop([categorical_attribute + '_?'], axis=1)
    except:
        print(categorical_attribute + '_? does not exist.')
```

```
    ## Replace original column with encoded columns
    df_adult = df_adult.drop([categorical_attribute], axis=1)
    df_adult = pd.concat([df_adult, one_hot], axis=1)
```

```
education_? does not exist.
marital-status_? does not exist.
relationship_? does not exist.
race_? does not exist.
sex_? does not exist.
salary_? does not exist.
```

[5]:
```
## Drop the target column 'salary_<=50K' because 'salary_>50K' alone is enough
df_adult = df_adult.drop(['salary_<=50K'], axis=1)
```

[6]:
```
## Normalize data in each column
x = df_adult.values
attributes = df_adult.columns.values
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
df_adult = pd.DataFrame(x_scaled)
df_adult.columns = attributes
```

[7]:
```
## Save cleaned dataset
df_adult.to_csv('datasets/cleaned/adult_cleaned.csv', index=False)
```

0.0.2 Clean Occupancy Dataset

https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+

[8]:
```
## Read dataset
df_occupancy_train = pd.read_csv('datasets/original/occupancy/occupancy_train.
 ↪csv')
df_occupancy_test1 = pd.read_csv('datasets/original/occupancy/occupancy_test1.
 ↪csv')
df_occupancy_test2 = pd.read_csv('datasets/original/occupancy/occupancy_test2.
 ↪csv')
df_occupancy = pd.concat([df_occupancy_train, df_occupancy_test1,␣
 ↪df_occupancy_test2], axis=0)
```

[9]:
```
## Drop 'data' column
df_occupancy = df_occupancy.drop(['date'], axis=1)
```

[10]:
```
## Normalize data in each column
x = df_occupancy.values
attributes = df_occupancy.columns.values
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
```

```
df_occupancy = pd.DataFrame(x_scaled)
df_occupancy.columns = attributes
```

[11]:
```
## Save cleaned dataset
df_occupancy.to_csv('datasets/cleaned/occupancy_cleaned.csv', index=False)
```

0.0.3 Clean HTRU2 Dataset

https://archive.ics.uci.edu/ml/datasets/HTRU2

[12]:
```
## Read dataset
df_HTRU2 = pd.read_csv('datasets/original/HTRU2_train.csv')
```

[13]:
```
## ip -> integrated profile; ds -> DM-SMR curve
df_HTRU2.columns = ['ip_mean', 'ip_standard_deviation', 'ip_excess_kurtosis',␣
 ↪'ip_skewness',
                    'ds_mean', 'ds_standard_deviation', 'ds_excess_kurtosis',␣
 ↪'ds_skewness', 'is_pulsar']
```

[14]:
```
## Normalize data in each column
x = df_HTRU2.values
attributes = df_HTRU2.columns.values
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
df_HTRU2 = pd.DataFrame(x_scaled)
df_HTRU2.columns = attributes
```

[15]:
```
## Save cleaned dataset
df_HTRU2.to_csv('datasets/cleaned/HTRU2_cleaned.csv', index=False)
```

0.0.4 Clean Activity Dataset

https://archive.ics.uci.edu/ml/datasets/Activity+recognition+with+healthy+older+people+using+a+batteryless

[16]:
```
## Read dataset

## Attribute encoding
#  time -> Time in seconds
#  acc_f -> Acceleration reading in G for frontal axis
#  acc_v -> Acceleration reading in G for vertical axis
#  acc_l -> Acceleration reading in G for lateral axis
#  sensor_id -> Id of antenna reading sensor
#  rssi -> Received signal strength indicator (RSSI)
#  phase -> Phase
#  frequency -> Frequency
#  activity -> Label of activity, 1: sit on bed, 2: sit on chair, 3: lying, 4:␣
 ↪ambulating
```

```
activity_dataset_path = 'datasets/original/activity/S1_Dataset'
activity_dataset_files = glob.glob(activity_dataset_path + "/*.csv")

dfs_activity = []

for activity_dataset_file in activity_dataset_files:
    df_activity = pd.read_csv(activity_dataset_file, header=None)
    dfs_activity.append(df_activity)

df_activity = pd.concat(dfs_activity, axis=0, ignore_index=True)
df_activity.columns = ['time', 'acc_f', 'acc_v', 'acc_l', 'sensor_id', 'rssi',␣
↪'phase', 'frequency', 'activity']
```

[17]:
```
## Drop 'time' column
df_activity = df_activity.drop(['time'], axis=1)
```

[18]:
```
## Apply one hot encoding to categorical attributes
categorical_attributes = ['sensor_id', 'activity']

for categorical_attribute in categorical_attributes:

    one_hot = pd.get_dummies(df_activity[categorical_attribute],␣
↪prefix=categorical_attribute)

    ## Replace original column with encoded columns
    df_activity = df_activity.drop([categorical_attribute], axis=1)
    df_activity = pd.concat([df_activity, one_hot], axis=1)
```

[19]:
```
## Drop 'activity_1', 'activity_2', and 'activity_3' to make this a binary␣
↪classification dataset.
##   And since we are more interested in whether the person is moving, we keep␣
↪'activity_4' as
##   the binary classification label.
df_activity = df_activity.drop(['activity_1', 'activity_2', 'activity_3'],␣
↪axis=1)

## Rename 'activity_4' to 'is_ambulating'
df_activity = df_activity.rename(columns={'activity_4': 'is_ambulating'})
```

[20]:
```
## Normalize data in each column
x = df_activity.values
attributes = df_activity.columns.values
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
df_activity = pd.DataFrame(x_scaled)
df_activity.columns = attributes
```

[21]:
```
## Save cleaned dataset
df_activity.to_csv('datasets/cleaned/activity_cleaned.csv', index=False)
```

model_training

March 17, 2021

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pickle
%config InlineBackend.figure_format = 'retina'
```

[3]:
```
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split, cross_val_score,␣
 ↪GridSearchCV, learning_curve, validation_curve
from sklearn.metrics import make_scorer, accuracy_score, f1_score,␣
 ↪roc_auc_score, matthews_corrcoef
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Perceptron
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
```

[4]:
```
## Read cleaned datasets
df_adult = pd.read_csv('datasets/cleaned/adult_cleaned.csv')
df_occupancy = pd.read_csv('datasets/cleaned/occupancy_cleaned.csv')
df_HTRU2 = pd.read_csv('datasets/cleaned/HTRU2_cleaned.csv')
df_activity = pd.read_csv('datasets/cleaned/activity_cleaned.csv')
dfs = [df_adult, df_occupancy, df_HTRU2, df_activity]
```

[24]:
```
print(df_adult.shape)
print(df_occupancy.shape)
print(df_HTRU2.shape)
print(df_activity.shape)
```

```
(32560, 106)
(20560, 6)
(17897, 9)
(52482, 11)
```

[25]:
```
print(df_adult.iloc[:, -1].value_counts())
print(df_occupancy.iloc[:, -1].value_counts())
print(df_HTRU2.iloc[:, -1].value_counts())
```

```
print(df_activity.iloc[:, -1].value_counts())
```

```
0.0    24719
1.0     7841
Name: salary_>50K, dtype: int64
0.0    15810
1.0     4750
Name: Occupancy, dtype: int64
0.0    16258
1.0     1639
Name: is_pulsar, dtype: int64
0.0    50526
1.0     1956
Name: is_ambulating, dtype: int64
```

[26]:
```
print(df_adult.iloc[:, -1].value_counts()[1] / sum(df_adult.iloc[:, -1].
↪value_counts()))
print(df_occupancy.iloc[:, -1].value_counts()[1] / sum(df_occupancy.iloc[:, -1].
↪value_counts()))
print(df_HTRU2.iloc[:, -1].value_counts()[1] / sum(df_HTRU2.iloc[:, -1].
↪value_counts()))
print(df_activity.iloc[:, -1].value_counts()[1] / sum(df_activity.iloc[:, -1].
↪value_counts()))
```

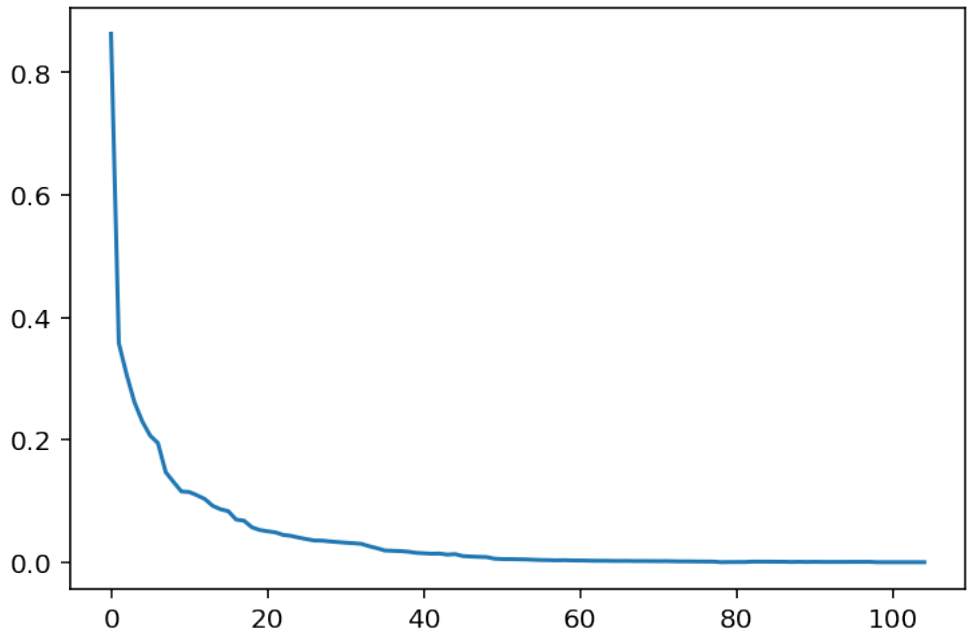
```
0.24081695331695332
0.23103112840466927
0.09157959434542103
0.037269921115811136
```

[4]:
```
## The Adult dataset contains 105 attributes. Here we seek to reduce its␣
↪dimensions
## Visualize the Adult dataset's principal components for dimensionality␣
↪reduction
X = df_adult.iloc[:, :-1]

## Compute covariance matrix of X
X_cov = np.cov(np.transpose(np.array(X)))

## Compute eigen decomposition of covariance matrix of X
eigenvalues, eigenvectors = np.linalg.eig(X_cov)

## Plot the eigenvalues in descending order
plt.plot(eigenvalues)

## From the plot below, we see that the top 30 principal components roughly␣
↪contains most of the information of
```

```
##  the Adult dataset, so we will map the Adult dataset to the subspace spanned␣
 ↪by its top 30 principal components.
pca_adult = PCA(n_components=30)
pca_adult.fit(X)

## We print the sum of explained variance ratio and see that the top 30␣
 ↪principal components explains 90% of the
##  variance, which confirms that using the top 30 principal components is a␣
 ↪good decision.
print(sum(pca_adult.explained_variance_ratio_))
```

```
C:\Users\Inffzy\anaconda3\lib\site-packages\numpy\core\_asarray.py:83:
ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```

```
0.9051441556184129
```



[5]:
```
## Prepare datasets
##   (We do 5 trials for each dataset, resulting in 20 splitted datasets in␣
 ↪total)

datasets = []

for i, df in enumerate(dfs):

    ## j is also used as random state control integer
```

```
    for j in range(5):
        X = df.iloc[:, :-1]
        Y = df.iloc[:, -1]

        X_train, X_test, Y_train, Y_test = train_test_split(X, Y,␣
↪train_size=5000, stratify=Y,
                                                            shuffle=True,␣
↪random_state=j)

        ## We use PCA to reduce the dimension of Adult dataset
        if i == 0:
            X_train = pca_adult.transform(X_train)
            X_test = pca_adult.transform(X_test)

        datasets.append([X_train, X_test, Y_train, Y_test])
```

[6]:
```
## Result storages
num_scores = 2    ## training [0] and testing [1] scores
num_models = 7    ## 7 variations of models
num_metrics = 3   ## accuracy, f1, roc auc
num_datasets = 20 ## 4 datasets * 5 trials

results_final = np.zeros((num_scores, num_models, num_metrics, num_datasets))
cv_results = []
```

[7]:
```
## Settings

models = [LogisticRegression(),
          Perceptron(),
          svm.SVC(),
          svm.SVC(),
          DecisionTreeClassifier(),
          RandomForestClassifier(),
          GradientBoostingClassifier()]

parameter_settings = [{'penalty': ['l2'],               ## Logistic Regression
                       'C': [1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3],
                       'solver': ['newton-cg']},

                      {'penalty': ['l1', 'l2'],         ## Perceptron
                       'alpha': [1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3]},

                      {'kernel': ['linear'],            ## SVM Linear
                       'C': [1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3]},

                      {'kernel': ['rbf'],               ## SVM RBF
                       'gamma': [1e-3, 1e-2, 1e-1, 1e0, 1e1],
```

```
                            'C': [1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3]},

                    {'criterion': ['gini', 'entropy'],  ## Decision Tree
                     'max_depth': [10, 50, 100, 500, 1000],
                     'ccp_alpha': [0, 1e-1, 1e0, 1e1]},

                    {'criterion': ['gini', 'entropy'],  ## Random Forest
                     'ccp_alpha': [0, 1e-1, 1e0, 1e1],
                  'n_estimators': [10, 50, 100, 500, 1000]},

                         {'loss': ['exponential'],      ## Gradient Boosting␣
↪Classifier
                  'learning_rate': [1e-3, 1e-2, 1e-1, 1e0, 1e1],
                   'n_estimators': [10, 50, 100, 500, 1000]}]

metrics = [accuracy_score, f1_score, roc_auc_score]
```

[8]:
```
for i, dataset in enumerate(datasets):

    X_train = datasets[i][0]
    Y_train = datasets[i][2]
    X_test = datasets[i][1]
    Y_test = datasets[i][3]

    for j, model in enumerate(models):
        for k, metric in enumerate(metrics):

            ## Make scorer for the current metric
            scorer = make_scorer(metric)

            ## Apply grid search with 5-fold cross validation
            classifier = GridSearchCV(model, parameter_settings[j],␣
↪scoring=scorer, cv=5, verbose=1)
            classifier.fit(X_train, Y_train)

            cv_results.append(classifier.cv_results_)

            ## Test the best trained parameters
            print('Best parameters on training data: ', classifier.best_params_)



            result_train = classifier.best_score_
            print('Training result: ', result_train)
            results_final[0, j, k, i] = result_train

            Y_pred = classifier.predict(X_test)
```

```
            result_test = metric(Y_test, Y_pred)
            print('Testing result: ', result_test)

            results_final[1, j, k, i] = result_test
```

```
Fitting 5 folds for each of 7 candidates, totalling 35 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  35 out of  35 | elapsed:    0.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

Best parameters on training data:  {'C': 1.0, 'penalty': 'l2', 'solver':
'newton-cg'}
Training result:  0.8298
Testing result:  0.8293541364296081
Fitting 5 folds for each of 7 candidates, totalling 35 fits

[Parallel(n_jobs=1)]: Done  35 out of  35 | elapsed:    0.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

Best parameters on training data:  {'C': 10.0, 'penalty': 'l2', 'solver':
'newton-cg'}
Training result:  0.6001144633307822
Testing result:  0.5981692189237745
Fitting 5 folds for each of 7 candidates, totalling 35 fits

[Parallel(n_jobs=1)]: Done  35 out of  35 | elapsed:    0.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

Best parameters on training data:  {'C': 10.0, 'penalty': 'l2', 'solver':
'newton-cg'}
Training result:  0.7281497278346671
Testing result:  0.7261880451889895
Fitting 5 folds for each of 14 candidates, totalling 70 fits

[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    0.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

Best parameters on training data:  {'alpha': 0.1, 'penalty': 'l2'}
Training result:  0.7626
Testing result:  0.6485849056603774
Fitting 5 folds for each of 14 candidates, totalling 70 fits

[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    0.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

Best parameters on training data:  {'alpha': 0.001, 'penalty': 'l1'}
Training result:  0.4940704959642317
Testing result:  0.6024663987806568
Fitting 5 folds for each of 14 candidates, totalling 70 fits

[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    0.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
Best parameters on training data:  {'learning_rate': 0.1, 'loss': 'exponential',
'n_estimators': 500}
Training result:  0.7266265597147951
Testing result:  0.7219128725843432
Fitting 5 folds for each of 25 candidates, totalling 125 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 125 out of 125 | elapsed:  2.1min finished

Best parameters on training data:  {'learning_rate': 1.0, 'loss': 'exponential',
'n_estimators': 100}
Training result:  0.8226194051329319
Testing result:  0.8256562704547658
```

[9]:
```
## Saving files
pickle.dump(cv_results, open('cv_results.p', 'wb'))
np.savez('results_final.npz', results_final)
```

[10]:
```
results_final.shape
```

[10]:
```
(2, 7, 3, 20)
```

result_visualization

March 18, 2021

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import pickle
from scipy import stats
%config InlineBackend.figure_format = 'retina'
```

[2]:
```
## Read model training results
cv_results = pickle.load(open('cv_results_3.p', 'rb'))
results_final = np.load('results_final_3.npz')['arr_0']
```

0.0.1 Use results_final to construct tables

[3]:
```
results_final.shape
```

[3]:
```
(2, 7, 3, 20)
```

[4]:
```
## Reshape results_final to make "trials" a separate dimension
## Resulting dimensions:
##  1D -> 2 {train = 0, test = 1}
##  2D -> 7 models
##  3D -> 3 metrics
##  4D -> 5 trials
##  5D -> 4 datasets

results_final_dataset_1 = results_final[:, :, :, 0:5].reshape(2, 7, 3, 5, 1)
results_final_dataset_2 = results_final[:, :, :, 5:10].reshape(2, 7, 3, 5, 1)
results_final_dataset_3 = results_final[:, :, :, 10:15].reshape(2, 7, 3, 5, 1)
results_final_dataset_4 = results_final[:, :, :, 15:20].reshape(2, 7, 3, 5, 1)
results_final = np.concatenate((results_final_dataset_1,␣
 ↪results_final_dataset_2, results_final_dataset_3, results_final_dataset_4),␣
 ↪axis=4)
results_final.shape
```

[4]:
```
(2, 7, 3, 5, 4)
```

[5]:
```
## Average results_final across trials to get a 2x7x3x4 numpy array
results_final_trial_avg = np.mean(results_final, axis = 3)
results_final_trial_avg.shape
```

[5]:
```
(2, 7, 3, 4)
```

[6]:
```
## Table 2: mean test set performance across trials for each algorithm/dataset
 ↪combo
##   shape = 7x(3+1) last column is the means
##   rows -> models
##   cols -> metrics
table_2 = np.mean(results_final_trial_avg[1, :, :, :], axis=2)
table_2_mean = np.mean(table_2, axis=1).reshape(7, 1)  ## Mean across metrics
table_2 = np.concatenate((table_2, table_2_mean), axis=1)

print(np.round(table_2, decimals=3))
```

```
[[0.943 0.757 0.834 0.845]
 [0.906 0.576 0.757 0.746]
 [0.943 0.767 0.838 0.849]
 [0.944 0.781 0.855 0.86 ]
 [0.936 0.773 0.864 0.858]
 [0.942 0.799 0.86  0.867]
 [0.946 0.802 0.868 0.872]]
```

[7]:
```
## Table 3: mean test set performance across trials for each algorithm/metric
 ↪combo
##   shape = 7x(4+1) last column is the means
##   rows -> models
##   cols -> datasets
table_3 = np.mean(results_final_trial_avg[1, :, :, :], axis=1)
table_3_mean = np.mean(table_3, axis=1).reshape(7, 1)  ## Mean across datasets
table_3 = np.concatenate((table_3, table_3_mean), axis=1)

print(np.round(table_3, decimals=3))
```

```
[[0.719 0.985 0.926 0.75  0.845]
 [0.649 0.928 0.857 0.552 0.746]
 [0.711 0.985 0.925 0.777 0.849]
 [0.716 0.986 0.925 0.813 0.86 ]
 [0.693 0.985 0.92  0.832 0.858]
 [0.7   0.988 0.927 0.853 0.867]
 [0.714 0.987 0.928 0.859 0.872]]
```

[8]:
```
## Table 4 (Appendix): mean training set performance across trials for each
 ↪algorithm/dataset combo
##   shape = 7x(3+1) last column is the means
```

```
##   rows -> models
##   cols -> metrics
table_4 = np.mean(results_final_trial_avg[0, :, :, :], axis=2)
table_4_mean = np.mean(table_4, axis=1).reshape(7, 1)  ## Mean across metrics
table_4 = np.concatenate((table_4, table_4_mean), axis=1)

print(np.round(table_4, decimals=3))
```

```
[[0.943 0.753 0.831 0.843]
 [0.918 0.623 0.78  0.773]
 [0.943 0.764 0.836 0.848]
 [0.944 0.777 0.851 0.857]
 [0.935 0.779 0.862 0.859]
 [0.941 0.795 0.858 0.865]
 [0.945 0.799 0.866 0.87 ]]
```

[9]:
```
## Table 5 (Appendix): raw testing set performance
##   Column 1: Adult dataset
print(np.round(results_final[1, :, :, 0, 0], decimals=3))  ## Adult dataset␣
 ↪trial 1
print(np.round(results_final[1, :, :, 1, 0], decimals=3))  ## Adult dataset␣
 ↪trial 2
print(np.round(results_final[1, :, :, 2, 0], decimals=3))  ## Adult dataset␣
 ↪trial 3
print(np.round(results_final[1, :, :, 3, 0], decimals=3))  ## Adult dataset␣
 ↪trial 4
print(np.round(results_final[1, :, :, 4, 0], decimals=3))  ## Adult dataset␣
 ↪trial 5
```

```
[[0.829 0.598 0.726]
 [0.649 0.602 0.745]
 [0.828 0.595 0.724]
 [0.828 0.601 0.729]
 [0.797 0.564 0.708]
 [0.812 0.568 0.711]
 [0.828 0.601 0.73 ]]
[[0.828 0.602 0.73 ]
 [0.787 0.531 0.693]
 [0.829 0.612 0.74 ]
 [0.828 0.61  0.736]
 [0.803 0.578 0.724]
 [0.809 0.577 0.718]
 [0.828 0.594 0.723]]
[[0.828 0.602 0.729]
 [0.771 0.412 0.62 ]
 [0.826 0.571 0.711]
 [0.826 0.599 0.727]
```

```
 [0.799 0.564 0.7  ]
 [0.813 0.58  0.715]
 [0.829 0.574 0.722]]
[[0.829 0.605 0.731]
 [0.766 0.585 0.735]
 [0.822 0.587 0.721]
 [0.826 0.6   0.729]
 [0.799 0.547 0.716]
 [0.814 0.573 0.712]
 [0.83  0.597 0.729]]
[[0.826 0.591 0.722]
 [0.792 0.414 0.627]
 [0.826 0.564 0.705]
 [0.827 0.566 0.707]
 [0.796 0.579 0.724]
 [0.812 0.572 0.714]
 [0.826 0.579 0.717]]
```

[10]:
```
##   Column 2: Occupancy dataset
print(np.round(results_final[1, :, :, 0, 1], decimals=3))  ## Occupancy dataset␣
 ↪trial 1
print(np.round(results_final[1, :, :, 1, 1], decimals=3))  ## Occupancy dataset␣
 ↪trial 2
print(np.round(results_final[1, :, :, 2, 1], decimals=3))  ## Occupancy dataset␣
 ↪trial 3
print(np.round(results_final[1, :, :, 3, 1], decimals=3))  ## Occupancy dataset␣
 ↪trial 4
print(np.round(results_final[1, :, :, 4, 1], decimals=3))  ## Occupancy dataset␣
 ↪trial 5
```

```
[[0.988 0.976 0.991]
 [0.985 0.968 0.989]
 [0.988 0.975 0.991]
 [0.99  0.978 0.993]
 [0.989 0.976 0.991]
 [0.991 0.981 0.991]
 [0.99  0.979 0.992]]
[[0.989 0.977 0.992]
 [0.988 0.975 0.988]
 [0.989 0.976 0.991]
 [0.99  0.979 0.992]
 [0.989 0.977 0.991]
 [0.99  0.98  0.99 ]
 [0.989 0.977 0.987]]
[[0.988 0.974 0.99 ]
 [0.913 0.777 0.822]
 [0.988 0.975 0.99 ]
```

```
 [0.988 0.974 0.992]
 [0.988 0.975 0.99 ]
 [0.991 0.981 0.99 ]
 [0.99  0.979 0.99 ]]
[[0.989 0.976 0.991]
 [0.902 0.825 0.936]
 [0.988 0.975 0.991]
 [0.988 0.976 0.991]
 [0.989 0.975 0.991]
 [0.992 0.981 0.991]
 [0.991 0.981 0.991]]
[[0.99  0.978 0.992]
 [0.978 0.894 0.985]
 [0.989 0.976 0.991]
 [0.991 0.98  0.993]
 [0.99  0.978 0.991]
 [0.992 0.982 0.991]
 [0.991 0.981 0.991]]
```

[11]:
```
##   Column 3: HTRU2 dataset
print(np.round(results_final[1, :, :, 0, 2], decimals=3))  ## HTRU2 dataset␣
 ↪trial 1
print(np.round(results_final[1, :, :, 1, 2], decimals=3))  ## HTRU2 dataset␣
 ↪trial 2
print(np.round(results_final[1, :, :, 2, 2], decimals=3))  ## HTRU2 dataset␣
 ↪trial 3
print(np.round(results_final[1, :, :, 3, 2], decimals=3))  ## HTRU2 dataset␣
 ↪trial 4
print(np.round(results_final[1, :, :, 4, 2], decimals=3))  ## HTRU2 dataset␣
 ↪trial 5
```

```
[[0.98  0.882 0.912]
 [0.976 0.855 0.891]
 [0.979 0.876 0.907]
 [0.98  0.886 0.915]
 [0.978 0.874 0.91 ]
 [0.98  0.886 0.922]
 [0.98  0.885 0.923]]
[[0.98  0.885 0.916]
 [0.977 0.868 0.762]
 [0.98  0.888 0.918]
 [0.979 0.877 0.91 ]
 [0.977 0.87  0.906]
 [0.98  0.882 0.918]
 [0.98  0.886 0.923]]
[[0.98  0.882 0.914]
 [0.975 0.852 0.856]
```

```
 [0.98  0.883 0.915]
 [0.979 0.875 0.906]
 [0.977 0.869 0.911]
 [0.979 0.88  0.91 ]
 [0.98  0.883 0.917]]
[[0.98  0.883 0.912]
 [0.947 0.598 0.714]
 [0.98  0.88  0.908]
 [0.98  0.891 0.917]
 [0.978 0.873 0.916]
 [0.98  0.886 0.917]
 [0.979 0.879 0.912]]
[[0.98  0.883 0.914]
 [0.967 0.783 0.826]
 [0.98  0.882 0.912]
 [0.98  0.882 0.916]
 [0.978 0.873 0.916]
 [0.979 0.883 0.92 ]
 [0.98  0.886 0.926]]
```

[12]:
```
##   Column 4: Activity dataset
print(np.round(results_final[1, :, :, 0, 3], decimals=3))  ## Activity dataset␣
 ↪trial 1
print(np.round(results_final[1, :, :, 1, 3], decimals=3))  ## Activity dataset␣
 ↪trial 2
print(np.round(results_final[1, :, :, 2, 3], decimals=3))  ## Activity dataset␣
 ↪trial 3
print(np.round(results_final[1, :, :, 3, 3], decimals=3))  ## Activity dataset␣
 ↪trial 4
print(np.round(results_final[1, :, :, 4, 3], decimals=3))  ## Activity dataset␣
 ↪trial 5
```

```
[[0.976 0.554 0.698]
 [0.885 0.025 0.506]
 [0.979 0.618 0.726]
 [0.98  0.667 0.784]
 [0.979 0.666 0.826]
 [0.983 0.731 0.801]
 [0.983 0.744 0.824]]
[[0.977 0.571 0.706]
 [0.963 0.058 0.515]
 [0.979 0.624 0.729]
 [0.979 0.683 0.79 ]
 [0.978 0.675 0.852]
 [0.984 0.771 0.829]
 [0.985 0.787 0.854]]
[[0.978 0.582 0.71 ]
```

```
 [0.963 0.353 0.69 ]
 [0.98  0.628 0.732]
 [0.98  0.662 0.782]
 [0.981 0.706 0.833]
 [0.986 0.771 0.822]
 [0.984 0.762 0.847]]
[[0.977 0.572 0.706]
 [0.97  0.13  0.736]
 [0.979 0.623 0.728]
 [0.979 0.677 0.801]
 [0.981 0.705 0.848]
 [0.986 0.774 0.81 ]
 [0.985 0.763 0.841]]
[[0.977 0.563 0.699]
 [0.967 0.018 0.5  ]
 [0.979 0.621 0.727]
 [0.978 0.657 0.801]
 [0.977 0.637 0.834]
 [0.983 0.747 0.821]
 [0.982 0.722 0.826]]
```

[13]:
```
## Table 6 (Appendix): pair-wise t-test for table 2 across 5 trials
##   shape = 7x3
##   rows -> models
##   cols -> metrics

table_6 = np.zeros((7, 3))
table_6_prep = np.mean(results_final[1, :, :, :, :], axis=3)

for i in range(3):  ## Loop through metrics

    idx_best = np.argmax(table_2[:, i])

    for j in range(7):  ## Loop through models

        p_val = stats.ttest_rel(table_6_prep[j, i, :], table_6_prep[idx_best,␣
↪i, :])[1]
        table_6[j, i] = p_val

print(np.round(table_6, decimals=3))
```

```
[[0.    0.    0.   ]
 [0.018 0.    0.   ]
 [0.01  0.    0.   ]
 [0.008 0.    0.001]
 [0.    0.    0.126]
 [0.    0.455 0.004]
 [  nan   nan   nan]]
```

[14]:
```
## Check which values we need to *
table_6 > 0.05
```

[14]:
```
array([[False, False, False],
       [False, False, False],
       [False, False, False],
       [False, False, False],
       [False, False,  True],
       [False,  True, False],
       [False, False, False]])
```

[15]:
```
## Table 7 (Appendix): pair-wise t-test for table 3 across 5 trials
##   shape = 7x4
##   rows -> models
##   cols -> datasets

table_7 = np.zeros((7, 4))
table_7_prep = np.mean(results_final[1, :, :, :, :], axis=1)

for i in range(4):  ## Loop through datasets

    idx_best = np.argmax(table_3[:, i])

    for j in range(7):  ## Loop through models

        p_val = stats.ttest_rel(table_7_prep[j, :, i], table_7_prep[idx_best, :
 ↪, i])[1]
        table_7[j, i] = p_val

print(np.round(table_7, decimals=3))
```

```
[[  nan 0.014 0.134 0.   ]
 [0.016 0.108 0.053 0.001]
 [0.157 0.003 0.131 0.   ]
 [0.449 0.139 0.293 0.001]
 [0.003 0.01  0.015 0.001]
 [0.      nan 0.522 0.184]
 [0.106 0.032   nan   nan]]
```

[16]:
```
## Check which values we need to *
table_7 > 0.05
```

[16]:
```
array([[False, False,  True, False],
       [False,  True,  True, False],
       [ True, False,  True, False],
       [ True,  True,  True, False],
       [False, False, False, False],
```

```
       [False, False,  True,  True],
       [ True, False, False, False]])
```

0.0.2 Use cv_results to draw heatmaps

420 = 20 x 7 (models) x 3 (metrics) 20 = 4 (datasets) x 5 (trials)

0 to 104: Adult dataset 105 to 209: Occupancy dataset 210 to 314: HTRU2 dataset 315 to 419: Activity dataset

Same model and same metric for every 21 cv_result

[17]:
```
len(cv_results)
```

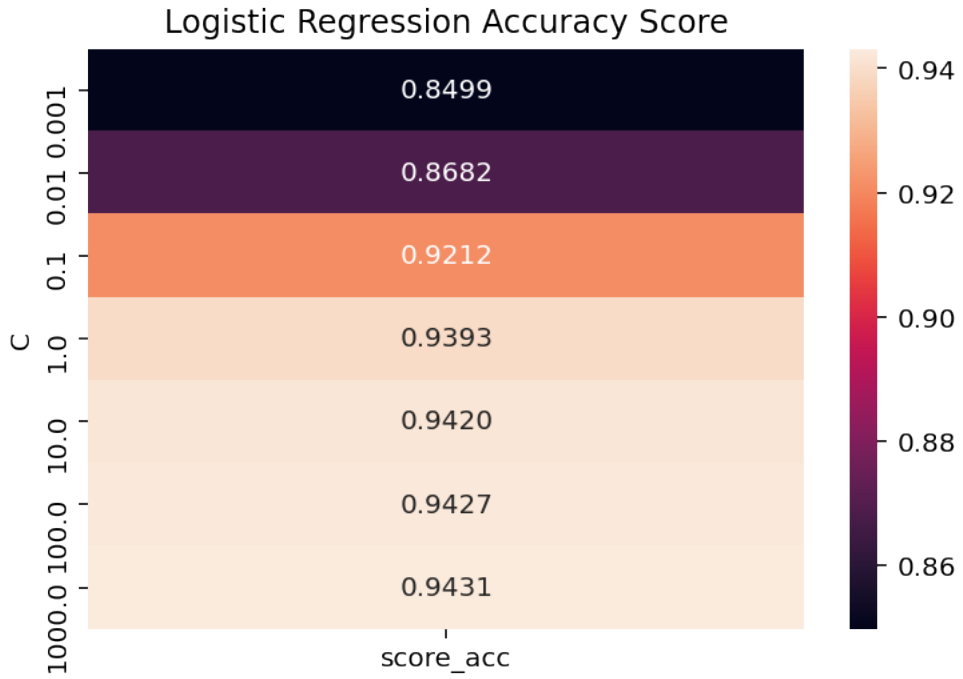
[17]:
```
420
```

[18]:
```
logreg_acc_idx =    list(np.arange(0, 420, 21))
logreg_f1_idx =     list(np.arange(1, 420, 21))
logreg_rocauc_idx = list(np.arange(2, 420, 21))

percep_acc_idx =    list(np.arange(3, 420, 21))
percep_f1_idx =     list(np.arange(4, 420, 21))
percep_rocauc_idx = list(np.arange(5, 420, 21))

linsvm_acc_idx =    list(np.arange(6, 420, 21))
linsvm_f1_idx =     list(np.arange(7, 420, 21))
linsvm_rocauc_idx = list(np.arange(8, 420, 21))

rbfsvm_acc_idx =    list(np.arange(9, 420, 21))
rbfsvm_f1_idx =     list(np.arange(10, 420, 21))
rbfsvm_rocauc_idx = list(np.arange(11, 420, 21))

dectre_acc_idx =    list(np.arange(12, 420, 21))
dectre_f1_idx =     list(np.arange(13, 420, 21))
dectre_rocauc_idx = list(np.arange(14, 420, 21))

ranfor_acc_idx =    list(np.arange(15, 420, 21))
ranfor_f1_idx =     list(np.arange(16, 420, 21))
ranfor_rocauc_idx = list(np.arange(17, 420, 21))

graboo_acc_idx =    list(np.arange(18, 420, 21))
graboo_f1_idx =     list(np.arange(19, 420, 21))
graboo_rocauc_idx = list(np.arange(20, 420, 21))
```

Logistic Regression

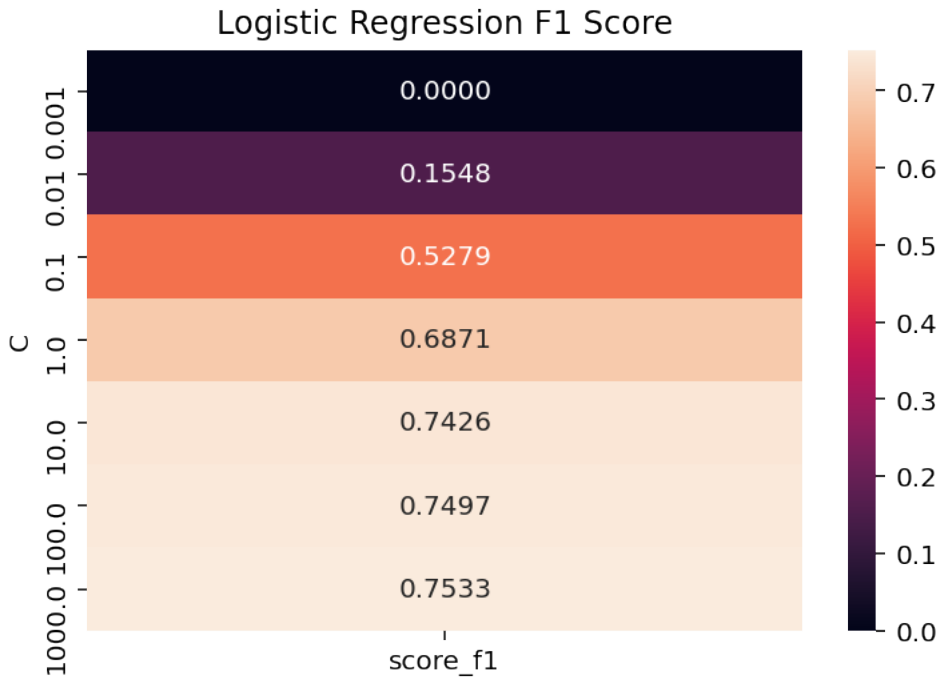
[19]:
```
## Accuracy
param_search_results_logreg_acc = pd.DataFrame()
```

```
for i in logreg_acc_idx:
    param_search_results = pd.DataFrame(cv_results[i]['params'])
    param_search_results['score_acc'] = cv_results[i]['mean_test_score']
    param_search_results_logreg_acc = pd.
 ↪concat([param_search_results_logreg_acc, param_search_results],␣
 ↪ignore_index=True)

param_search_results_logreg_acc = param_search_results_logreg_acc.
 ↪groupby(['C'], as_index=True).mean()
sns.heatmap(param_search_results_logreg_acc, annot=True, fmt='.4f')
plt.title('Logistic Regression Accuracy Score')
plt.savefig('results/param_search_results_logreg_acc.png')
plt.show()
```

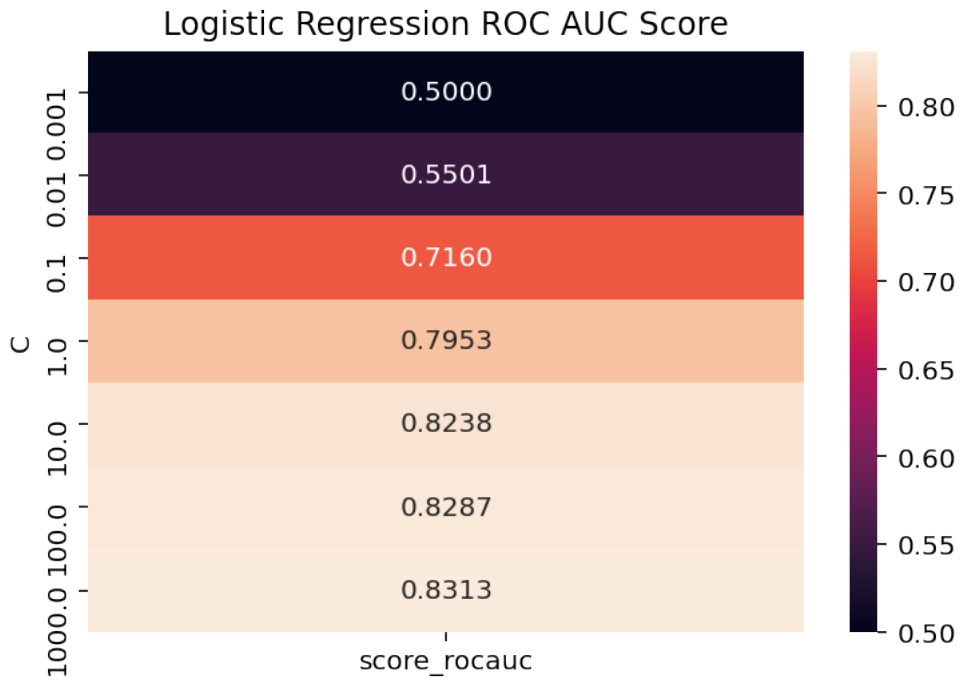


[20]:
```
## F1
param_search_results_logreg_f1 = pd.DataFrame()

for i in logreg_f1_idx:
    param_search_results = pd.DataFrame(cv_results[i]['params'])
    param_search_results['score_f1'] = cv_results[i]['mean_test_score']
    param_search_results_logreg_f1 = pd.concat([param_search_results_logreg_f1,␣
 ↪param_search_results], ignore_index=True)
```

```
param_search_results_logreg_f1 = param_search_results_logreg_f1.groupby(['C'],␣
↪as_index=True).mean()
sns.heatmap(param_search_results_logreg_f1, annot=True, fmt='.4f')
plt.title('Logistic Regression F1 Score')
plt.savefig('results/param_search_results_logreg_f1.png')
plt.show()
```

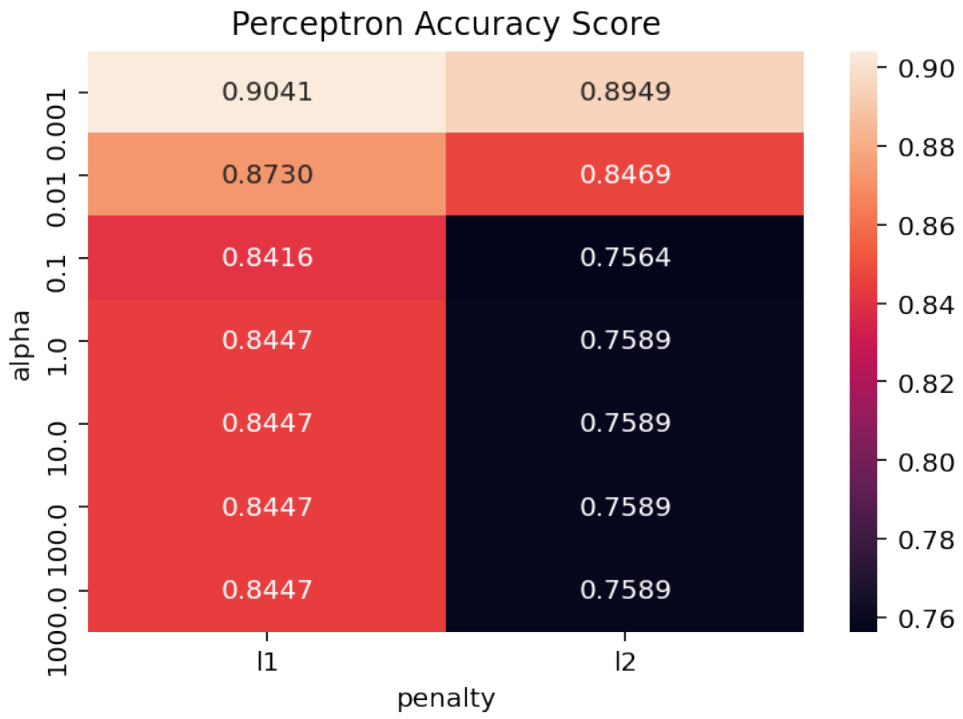


[21]:
```
## ROC AUC
param_search_results_logreg_rocauc = pd.DataFrame()

for i in logreg_rocauc_idx:
    param_search_results = pd.DataFrame(cv_results[i]['params'])
    param_search_results['score_rocauc'] = cv_results[i]['mean_test_score']
    param_search_results_logreg_rocauc = pd.
↪concat([param_search_results_logreg_rocauc, param_search_results],␣
↪ignore_index=True)

param_search_results_logreg_rocauc = param_search_results_logreg_rocauc.
↪groupby(['C'], as_index=True).mean()
sns.heatmap(param_search_results_logreg_rocauc, annot=True, fmt='.4f')
plt.title('Logistic Regression ROC AUC Score')
plt.savefig('results/param_search_results_logreg_rocauc.png')
plt.show()
```

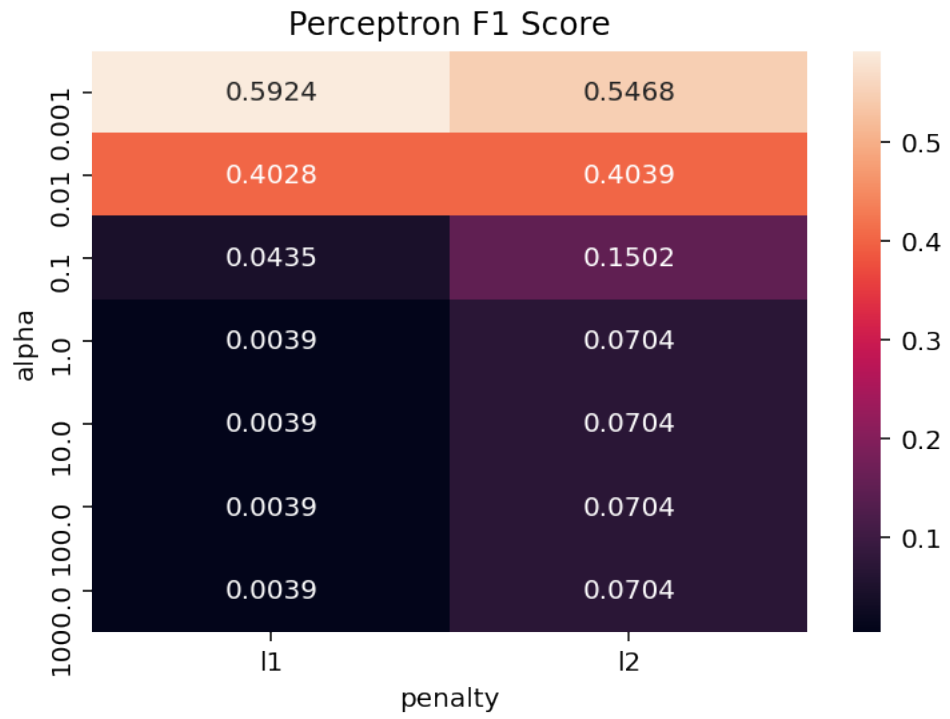


Perceptron

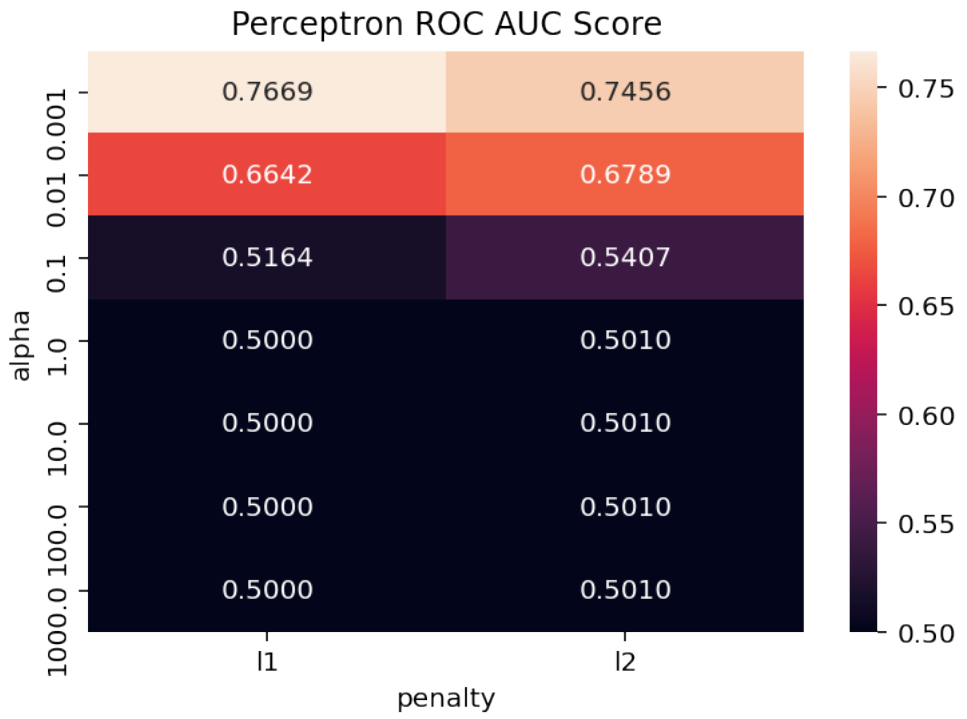
[22]:
```
## Accuracy
param_search_results_percep_acc = pd.DataFrame()

for i in percep_acc_idx:
    param_search_results = pd.DataFrame(cv_results[i]['params'])
    param_search_results['score_acc'] = cv_results[i]['mean_test_score']
    param_search_results_percep_acc = pd.
 ↪concat([param_search_results_percep_acc, param_search_results],␣
 ↪ignore_index=True)

param_search_results_percep_acc = param_search_results_percep_acc.
 ↪groupby(['penalty', 'alpha'], as_index=True, group_keys=False).mean()
param_search_results_percep_acc.reset_index(inplace=True)
sns.heatmap(param_search_results_percep_acc.pivot('alpha', 'penalty',␣
 ↪'score_acc'), annot=True, fmt='.4f')
plt.title('Perceptron Accuracy Score')
plt.xlabel('penalty')
plt.savefig('results/param_search_results_percep_acc.png')
plt.show()
```



[23]:
```
## F1
param_search_results_percep_f1 = pd.DataFrame()

for i in percep_f1_idx:
    param_search_results = pd.DataFrame(cv_results[i]['params'])
    param_search_results['score_f1'] = cv_results[i]['mean_test_score']
    param_search_results_percep_f1 = pd.concat([param_search_results_percep_f1,␣
 ↪param_search_results], ignore_index=True)

param_search_results_percep_f1 = param_search_results_percep_f1.
 ↪groupby(['penalty', 'alpha'], as_index=True, group_keys=False).mean()
param_search_results_percep_f1.reset_index(inplace=True)
sns.heatmap(param_search_results_percep_f1.pivot('alpha', 'penalty',␣
 ↪'score_f1'), annot=True, fmt='.4f')
plt.title('Perceptron F1 Score')
plt.xlabel('penalty')
plt.savefig('results/param_search_results_percep_f1.png')
plt.show()
```

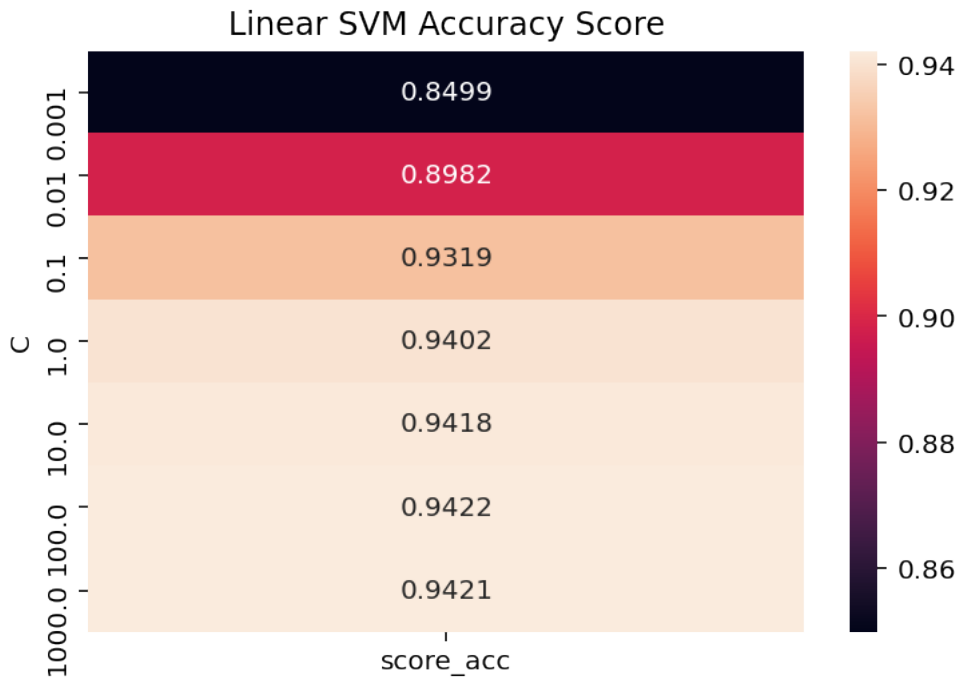


[24]:
```
## ROC AUC
param_search_results_percep_rocauc = pd.DataFrame()

for i in percep_rocauc_idx:
    param_search_results = pd.DataFrame(cv_results[i]['params'])
    param_search_results['score_rocauc'] = cv_results[i]['mean_test_score']
    param_search_results_percep_rocauc = pd.
 ↪concat([param_search_results_percep_rocauc, param_search_results],␣
 ↪ignore_index=True)

param_search_results_percep_rocauc = param_search_results_percep_rocauc.
 ↪groupby(['penalty', 'alpha'], as_index=True, group_keys=False).mean()
param_search_results_percep_rocauc.reset_index(inplace=True)
sns.heatmap(param_search_results_percep_rocauc.pivot('alpha', 'penalty',␣
 ↪'score_rocauc'), annot=True, fmt='.4f')
plt.title('Perceptron ROC AUC Score')
plt.xlabel('penalty')
plt.savefig('results/param_search_results_percep_rocauc.png')
plt.show()
```

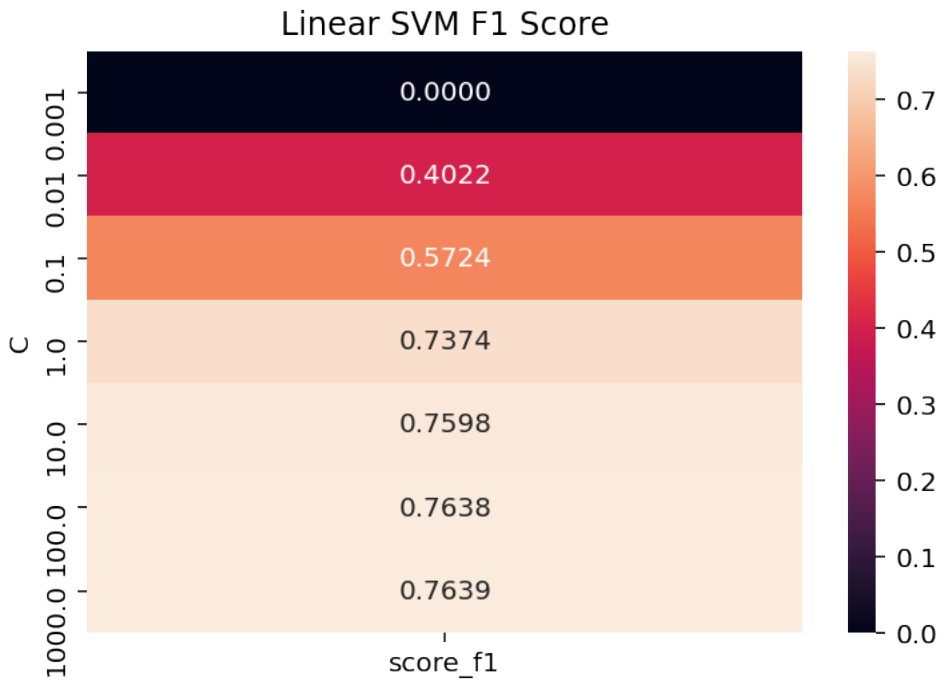


Linear SVM

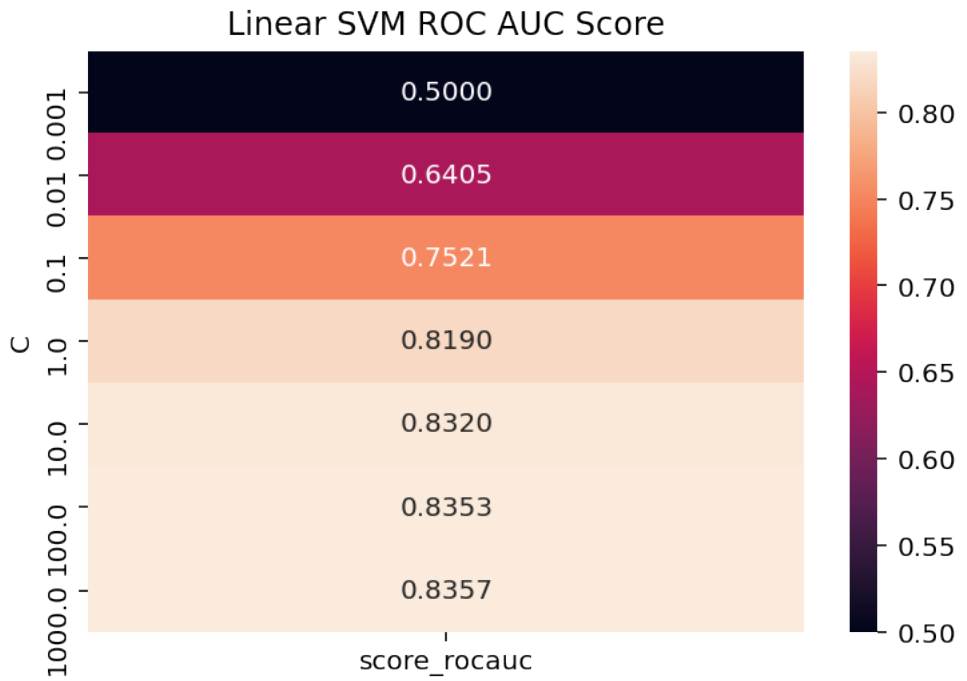
[25]:
```
## Accuracy
param_search_results_linsvm_acc = pd.DataFrame()

for i in linsvm_acc_idx:
    param_search_results = pd.DataFrame(cv_results[i]['params'])
    param_search_results['score_acc'] = cv_results[i]['mean_test_score']
    param_search_results_linsvm_acc = pd.
 ↪concat([param_search_results_linsvm_acc, param_search_results],␣
 ↪ignore_index=True)

param_search_results_linsvm_acc = param_search_results_linsvm_acc.
 ↪groupby(['C'], as_index=True).mean()
sns.heatmap(param_search_results_linsvm_acc, annot=True, fmt='.4f')
plt.title('Linear SVM Accuracy Score')
plt.savefig('results/param_search_results_linsvm_acc.png')
plt.show()
```



[26]:
```
## F1
param_search_results_linsvm_f1 = pd.DataFrame()

for i in linsvm_f1_idx:
    param_search_results = pd.DataFrame(cv_results[i]['params'])
    param_search_results['score_f1'] = cv_results[i]['mean_test_score']
    param_search_results_linsvm_f1 = pd.concat([param_search_results_linsvm_f1,␣
 ↪param_search_results], ignore_index=True)

param_search_results_linsvm_f1 = param_search_results_linsvm_f1.groupby(['C'],␣
 ↪as_index=True).mean()
sns.heatmap(param_search_results_linsvm_f1, annot=True, fmt='.4f')
plt.title('Linear SVM F1 Score')
plt.savefig('results/param_search_results_linsvm_f1.png')
plt.show()
```

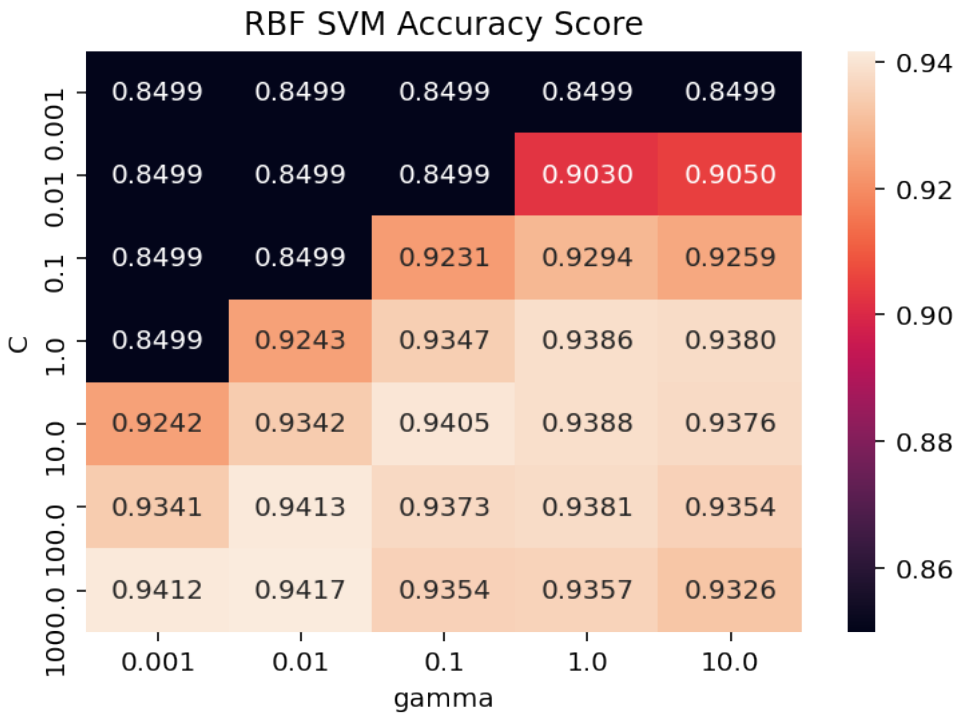


[27]:
```
## ROC AUC
param_search_results_linsvm_rocauc = pd.DataFrame()

for i in linsvm_rocauc_idx:
    param_search_results = pd.DataFrame(cv_results[i]['params'])
    param_search_results['score_rocauc'] = cv_results[i]['mean_test_score']
    param_search_results_linsvm_rocauc = pd.
 ↪concat([param_search_results_linsvm_rocauc, param_search_results],␣
 ↪ignore_index=True)

param_search_results_linsvm_rocauc = param_search_results_linsvm_rocauc.
 ↪groupby(['C'], as_index=True).mean()
sns.heatmap(param_search_results_linsvm_rocauc, annot=True, fmt='.4f')
plt.title('Linear SVM ROC AUC Score')
plt.savefig('results/param_search_results_linsvm_rocauc.png')
plt.show()
```

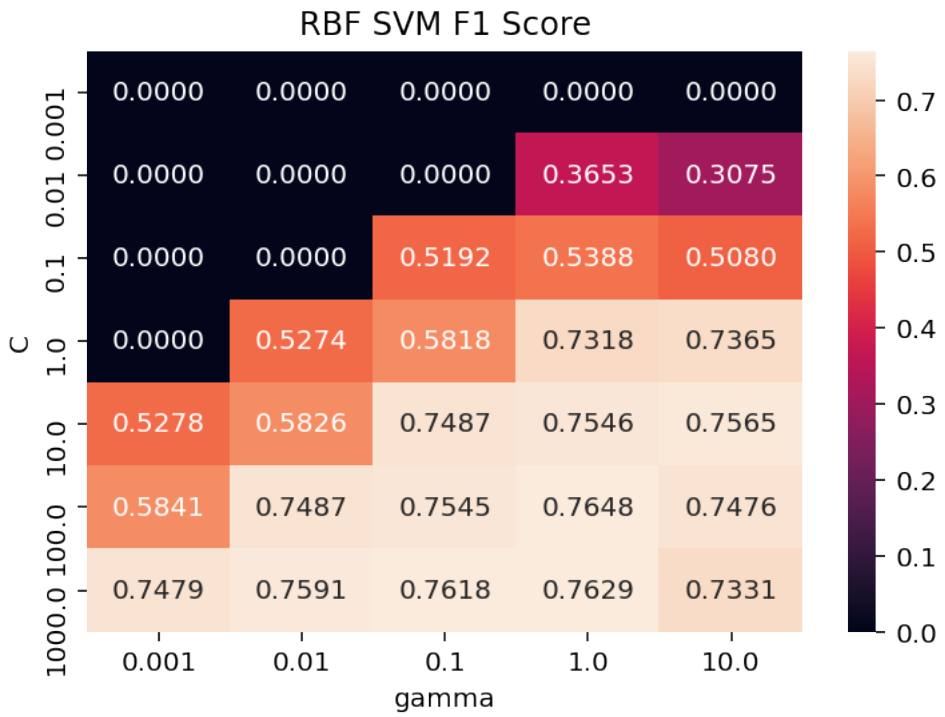


RBF SVM

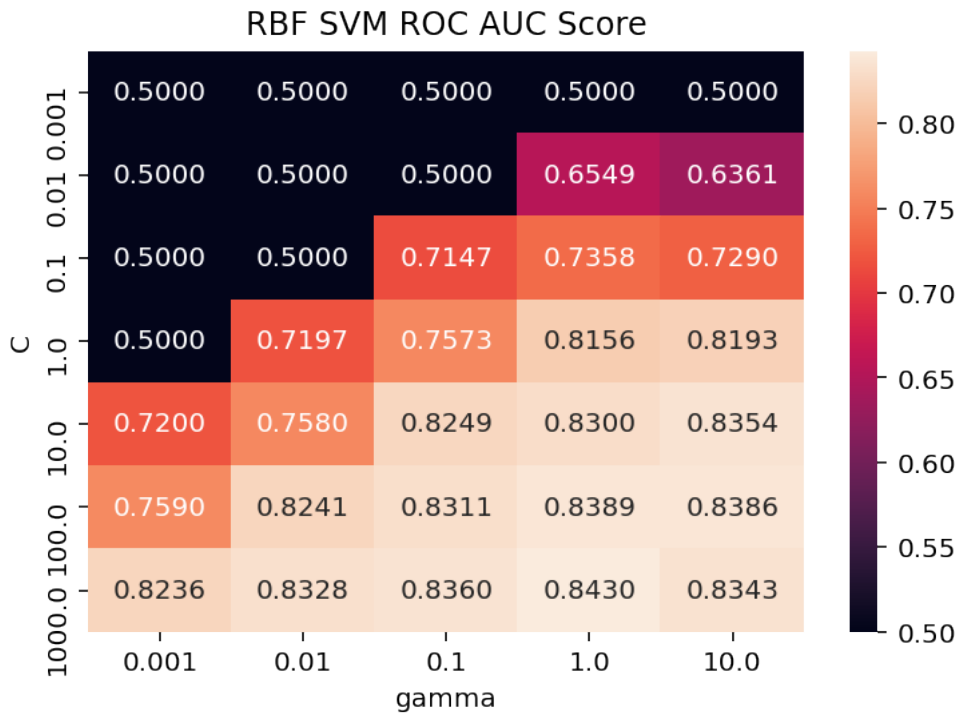
[28]:
```
## Accuracy
param_search_results_rbfsvm_acc = pd.DataFrame()

for i in rbfsvm_acc_idx:
    param_search_results = pd.DataFrame(cv_results[i]['params'])
    param_search_results['score_acc'] = cv_results[i]['mean_test_score']
    param_search_results_rbfsvm_acc = pd.
 ↪concat([param_search_results_rbfsvm_acc, param_search_results],␣
 ↪ignore_index=True)

param_search_results_rbfsvm_acc = param_search_results_rbfsvm_acc.groupby(['C',␣
 ↪'gamma'], as_index=True, group_keys=False).mean()
param_search_results_rbfsvm_acc.reset_index(inplace=True)
sns.heatmap(param_search_results_rbfsvm_acc.pivot('C', 'gamma', 'score_acc'),␣
 ↪annot=True, fmt='.4f')
plt.xlabel('gamma')
plt.title('RBF SVM Accuracy Score')
plt.savefig('results/param_search_results_rbfsvm_acc.png')
plt.show()
```



[29]:
```
## F1
param_search_results_rbfsvm_f1 = pd.DataFrame()

for i in rbfsvm_f1_idx:
    param_search_results = pd.DataFrame(cv_results[i]['params'])
    param_search_results['score_f1'] = cv_results[i]['mean_test_score']
    param_search_results_rbfsvm_f1 = pd.concat([param_search_results_rbfsvm_f1,␣
 ↪param_search_results], ignore_index=True)

param_search_results_rbfsvm_f1 = param_search_results_rbfsvm_f1.groupby(['C',␣
 ↪'gamma'], as_index=True, group_keys=False).mean()
param_search_results_rbfsvm_f1.reset_index(inplace=True)
sns.heatmap(param_search_results_rbfsvm_f1.pivot('C', 'gamma', 'score_f1'),␣
 ↪annot=True, fmt='.4f')
plt.xlabel('gamma')
plt.title('RBF SVM F1 Score')
plt.savefig('results/param_search_results_rbfsvm_f1.png')
plt.show()
```



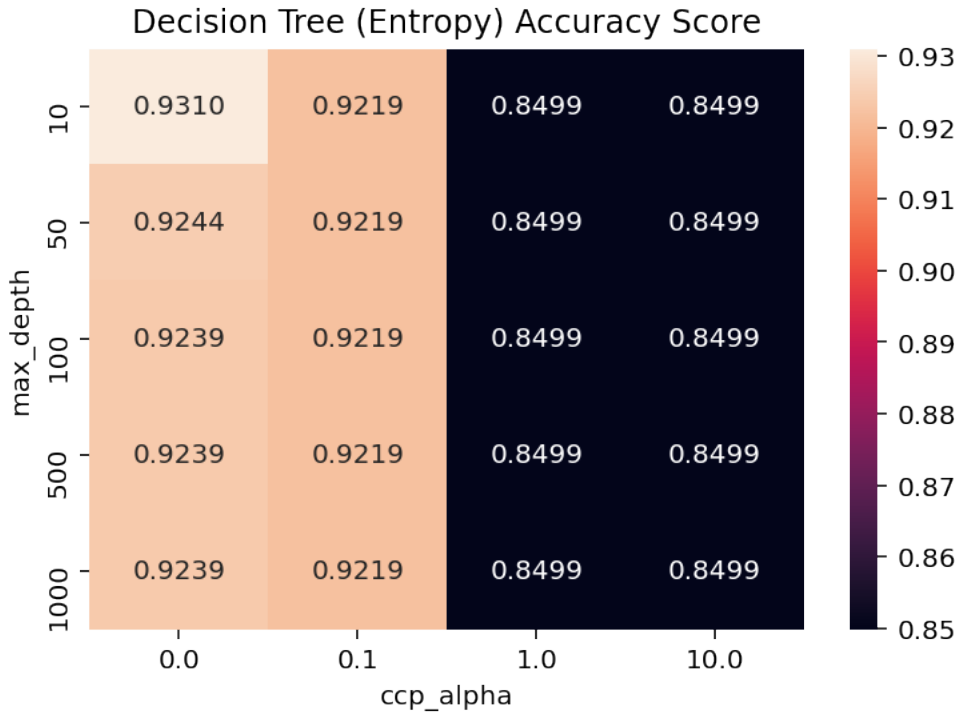
[30]:
```
## ROC AUC
param_search_results_rbfsvm_rocauc = pd.DataFrame()

for i in rbfsvm_rocauc_idx:
    param_search_results = pd.DataFrame(cv_results[i]['params'])
    param_search_results['score_rocauc'] = cv_results[i]['mean_test_score']
    param_search_results_rbfsvm_rocauc = pd.
 ↪concat([param_search_results_rbfsvm_rocauc, param_search_results],␣
 ↪ignore_index=True)

param_search_results_rbfsvm_rocauc = param_search_results_rbfsvm_rocauc.
 ↪groupby(['C', 'gamma'], as_index=True, group_keys=False).mean()
param_search_results_rbfsvm_rocauc.reset_index(inplace=True)
sns.heatmap(param_search_results_rbfsvm_rocauc.pivot('C', 'gamma',␣
 ↪'score_rocauc'), annot=True, fmt='.4f')
plt.xlabel('gamma')
plt.title('RBF SVM ROC AUC Score')
plt.savefig('results/param_search_results_rbfsvm_rocauc.png')
plt.show()
```

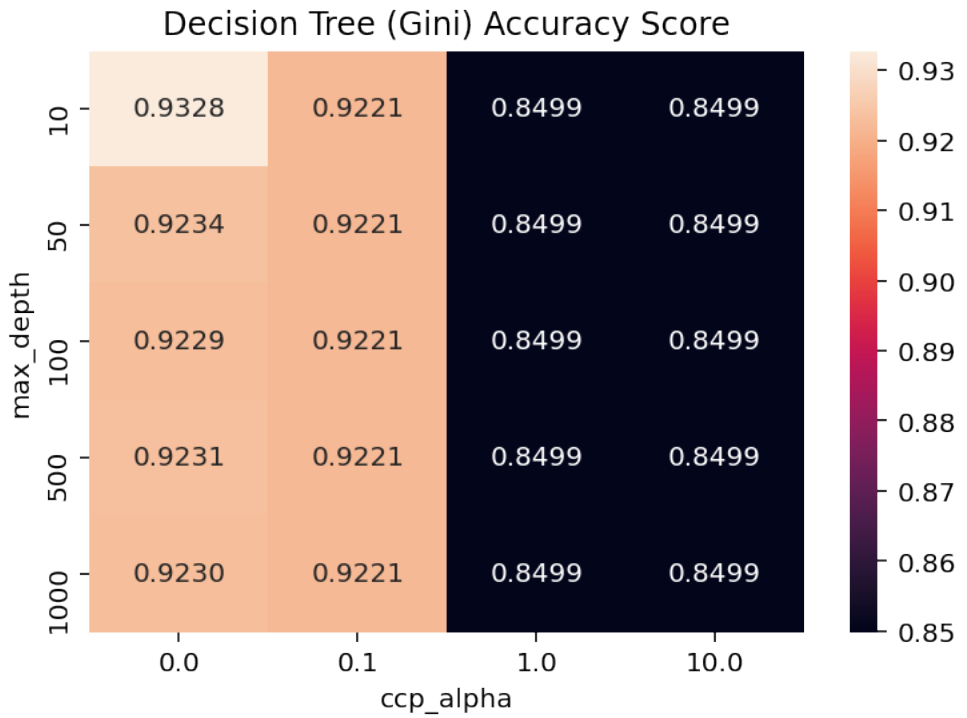


Decision Tree

[31]:
```
## Accuracy
param_search_results_dectre_acc = pd.DataFrame()

for i in dectre_acc_idx:
    param_search_results = pd.DataFrame(cv_results[i]['params'])
    param_search_results['score_acc'] = cv_results[i]['mean_test_score']
    param_search_results_dectre_acc = pd.
 ↪concat([param_search_results_dectre_acc, param_search_results],␣
 ↪ignore_index=True)

param_search_results_dectre_acc = param_search_results_dectre_acc.
 ↪groupby(['max_depth', 'ccp_alpha', 'criterion'], as_index=True,␣
 ↪group_keys=False).mean()
param_search_results_dectre_acc.reset_index(inplace=True)

param_search_results_dectre_acc_entropy =␣
 ↪param_search_results_dectre_acc[param_search_results_dectre_acc['criterion']␣
 ↪== 'entropy']
param_search_results_dectre_acc_entropy =␣
 ↪param_search_results_dectre_acc_entropy.drop(columns=['criterion'])

plt.figure()
```

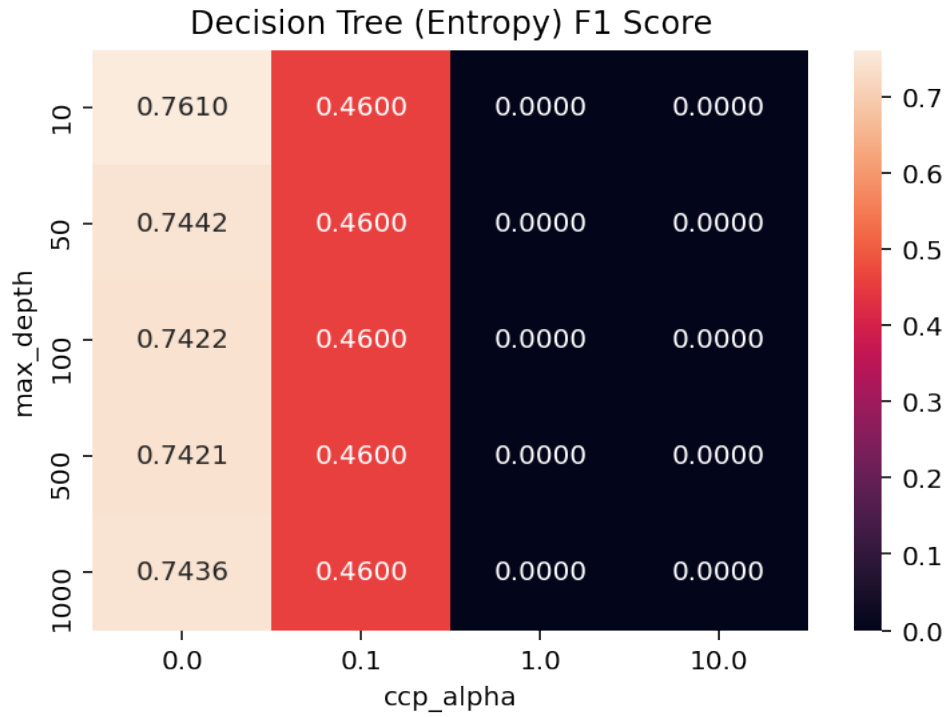
```
sns.heatmap(param_search_results_dectre_acc_entropy.pivot('max_depth',␣
 ↪'ccp_alpha', 'score_acc'), annot=True, fmt='.4f')
plt.xlabel('ccp_alpha')
plt.title('Decision Tree (Entropy) Accuracy Score')
plt.savefig('results/param_search_results_dectre_entropy_acc.png')
plt.show()

param_search_results_dectre_acc_gini =␣
 ↪param_search_results_dectre_acc[param_search_results_dectre_acc['criterion']␣
 ↪== 'gini']
param_search_results_dectre_acc_gini = param_search_results_dectre_acc_gini.
 ↪drop(columns=['criterion'])

plt.figure()
sns.heatmap(param_search_results_dectre_acc_gini.pivot('max_depth',␣
 ↪'ccp_alpha', 'score_acc'), annot=True, fmt='.4f')
plt.xlabel('ccp_alpha')
plt.title('Decision Tree (Gini) Accuracy Score')
plt.savefig('results/param_search_results_dectre_gini_acc.png')
plt.show()
```

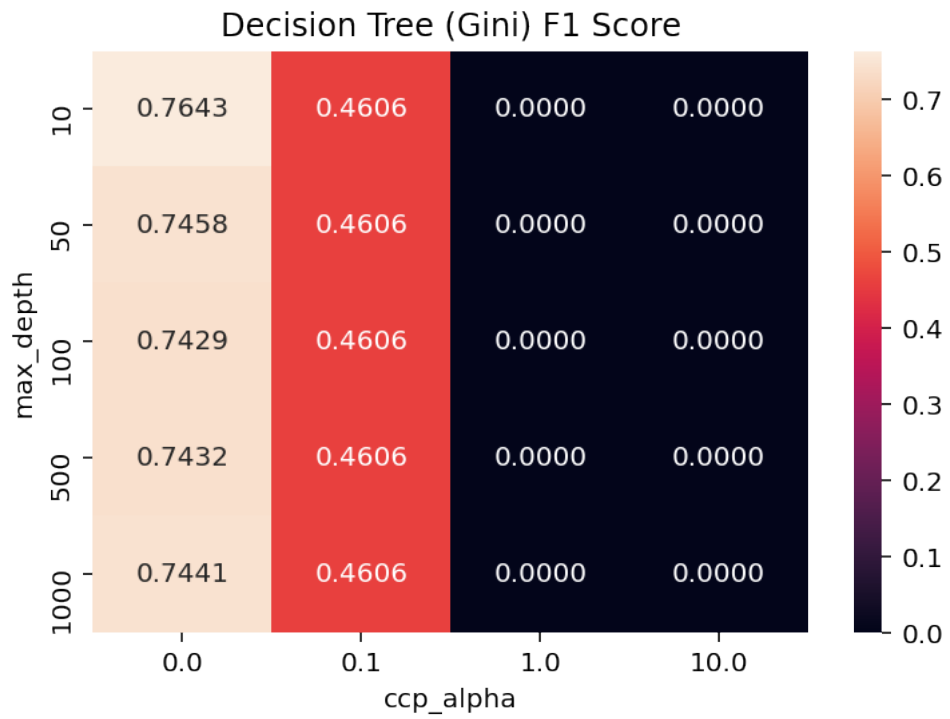




[32]:
```
## F1
param_search_results_dectre_f1 = pd.DataFrame()

for i in dectre_f1_idx:
    param_search_results = pd.DataFrame(cv_results[i]['params'])
    param_search_results['score_f1'] = cv_results[i]['mean_test_score']
    param_search_results_dectre_f1 = pd.concat([param_search_results_dectre_f1,␣
 ↪param_search_results], ignore_index=True)

param_search_results_dectre_f1 = param_search_results_dectre_f1.
 ↪groupby(['max_depth', 'ccp_alpha', 'criterion'], as_index=True,␣
 ↪group_keys=False).mean()
param_search_results_dectre_f1.reset_index(inplace=True)

param_search_results_dectre_f1_entropy =␣
 ↪param_search_results_dectre_f1[param_search_results_dectre_f1['criterion']␣
 ↪== 'entropy']
param_search_results_dectre_f1_entropy = param_search_results_dectre_f1_entropy.
 ↪drop(columns=['criterion'])

plt.figure()
sns.heatmap(param_search_results_dectre_f1_entropy.pivot('max_depth',␣
 ↪'ccp_alpha', 'score_f1'), annot=True, fmt='.4f')
```

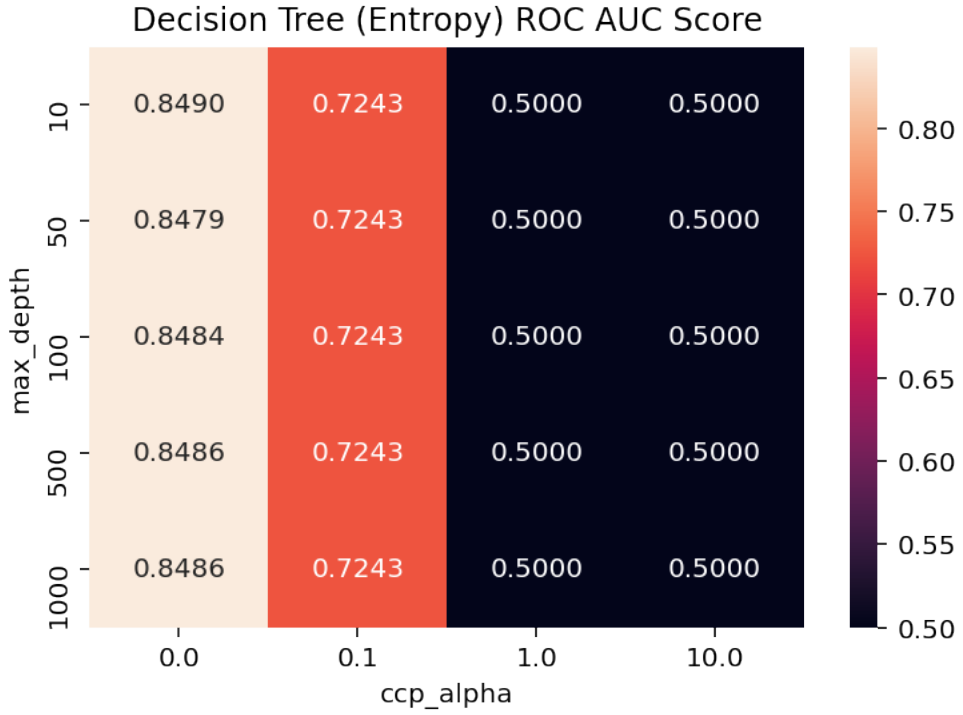
```
plt.xlabel('ccp_alpha')
plt.title('Decision Tree (Entropy) F1 Score')
plt.savefig('results/param_search_results_dectre_entropy_f1.png')
plt.show()

param_search_results_dectre_f1_gini =␣
↪param_search_results_dectre_f1[param_search_results_dectre_f1['criterion']␣
↪== 'gini']
param_search_results_dectre_f1_gini = param_search_results_dectre_f1_gini.
↪drop(columns=['criterion'])

plt.figure()
sns.heatmap(param_search_results_dectre_f1_gini.pivot('max_depth', 'ccp_alpha',␣
↪'score_f1'), annot=True, fmt='.4f')
plt.xlabel('ccp_alpha')
plt.title('Decision Tree (Gini) F1 Score')
plt.savefig('results/param_search_results_dectre_gini_f1.png')
plt.show()
```

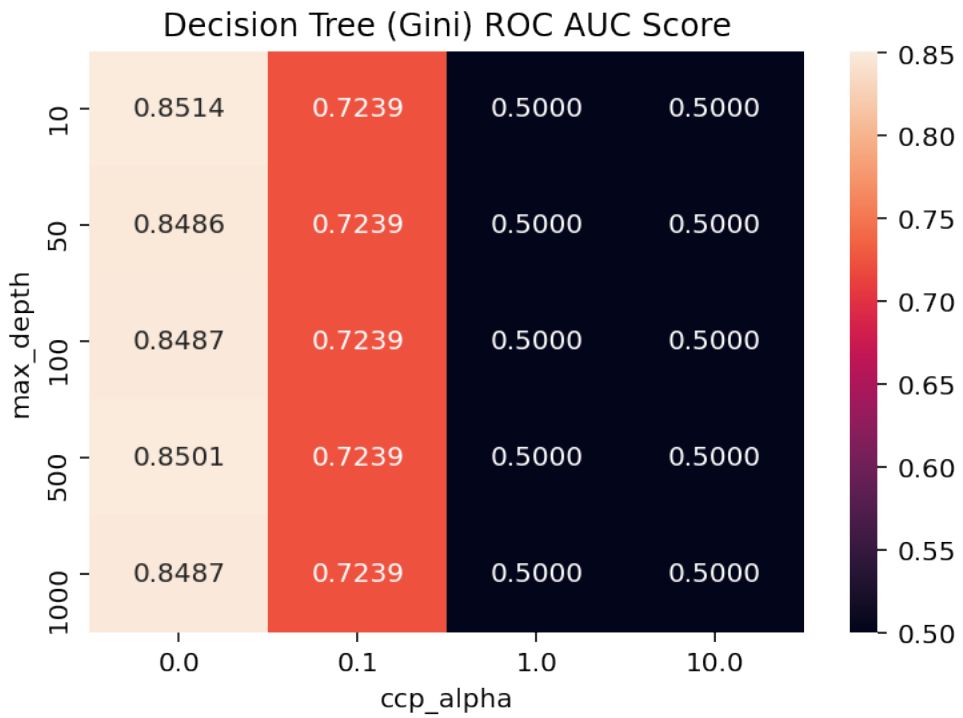




[33]:
```
## ROC AUC
param_search_results_dectre_rocauc = pd.DataFrame()

for i in dectre_rocauc_idx:
    param_search_results = pd.DataFrame(cv_results[i]['params'])
    param_search_results['score_rocauc'] = cv_results[i]['mean_test_score']
    param_search_results_dectre_rocauc = pd.
 ↪concat([param_search_results_dectre_rocauc, param_search_results],␣
 ↪ignore_index=True)

param_search_results_dectre_rocauc = param_search_results_dectre_rocauc.
 ↪groupby(['max_depth', 'ccp_alpha', 'criterion'], as_index=True,␣
 ↪group_keys=False).mean()
param_search_results_dectre_rocauc.reset_index(inplace=True)

param_search_results_dectre_rocauc_entropy =␣
 ↪param_search_results_dectre_rocauc[param_search_results_dectre_rocauc['criterion']␣
 ↪== 'entropy']
param_search_results_dectre_rocauc_entropy =␣
 ↪param_search_results_dectre_rocauc_entropy.drop(columns=['criterion'])

plt.figure()
```

```
sns.heatmap(param_search_results_dectre_rocauc_entropy.pivot('max_depth',␣
↪'ccp_alpha', 'score_rocauc'), annot=True, fmt='.4f')
plt.xlabel('ccp_alpha')
plt.title('Decision Tree (Entropy) ROC AUC Score')
plt.savefig('results/param_search_results_dectre_entropy_rocauc.png')
plt.show()

param_search_results_dectre_rocauc_gini =␣
↪param_search_results_dectre_rocauc[param_search_results_dectre_rocauc['criterion']␣
↪== 'gini']
param_search_results_dectre_rocauc_gini =␣
↪param_search_results_dectre_rocauc_gini.drop(columns=['criterion'])

plt.figure()
sns.heatmap(param_search_results_dectre_rocauc_gini.pivot('max_depth',␣
↪'ccp_alpha', 'score_rocauc'), annot=True, fmt='.4f')
plt.xlabel('ccp_alpha')
plt.title('Decision Tree (Gini) ROC AUC Score')
plt.savefig('results/param_search_results_dectre_gini_rocauc.png')
plt.show()
```

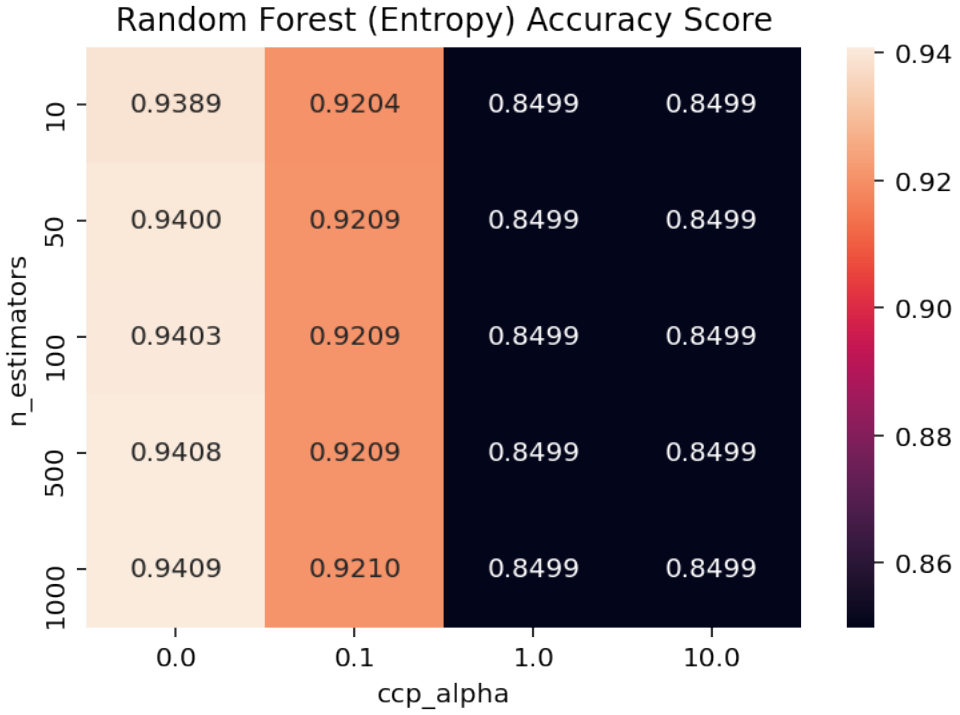


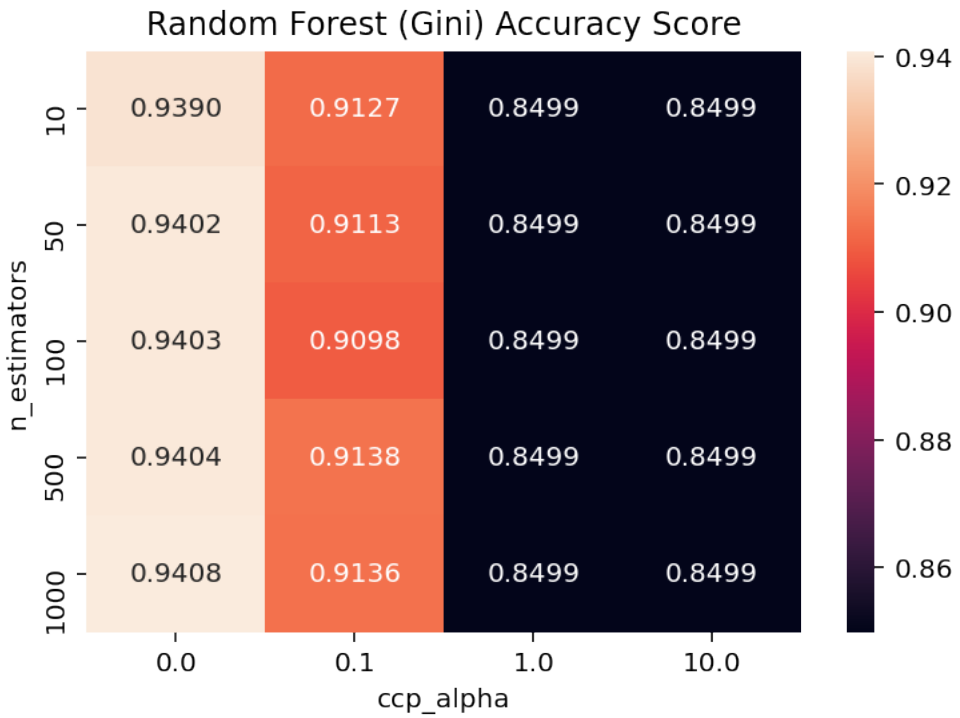


Random Forest

[34]:
```
## Accuracy
param_search_results_ranfor_acc = pd.DataFrame()

for i in ranfor_acc_idx:
    param_search_results = pd.DataFrame(cv_results[i]['params'])
    param_search_results['score_acc'] = cv_results[i]['mean_test_score']
    param_search_results_ranfor_acc = pd.
 ↪concat([param_search_results_ranfor_acc, param_search_results],␣
 ↪ignore_index=True)

param_search_results_ranfor_acc = param_search_results_ranfor_acc.
 ↪groupby(['n_estimators', 'ccp_alpha', 'criterion'], as_index=True,␣
 ↪group_keys=False).mean()
param_search_results_ranfor_acc.reset_index(inplace=True)

param_search_results_ranfor_acc_entropy =␣
 ↪param_search_results_ranfor_acc[param_search_results_ranfor_acc['criterion']␣
 ↪== 'entropy']
param_search_results_ranfor_acc_entropy =␣
 ↪param_search_results_ranfor_acc_entropy.drop(columns=['criterion'])

plt.figure()
```

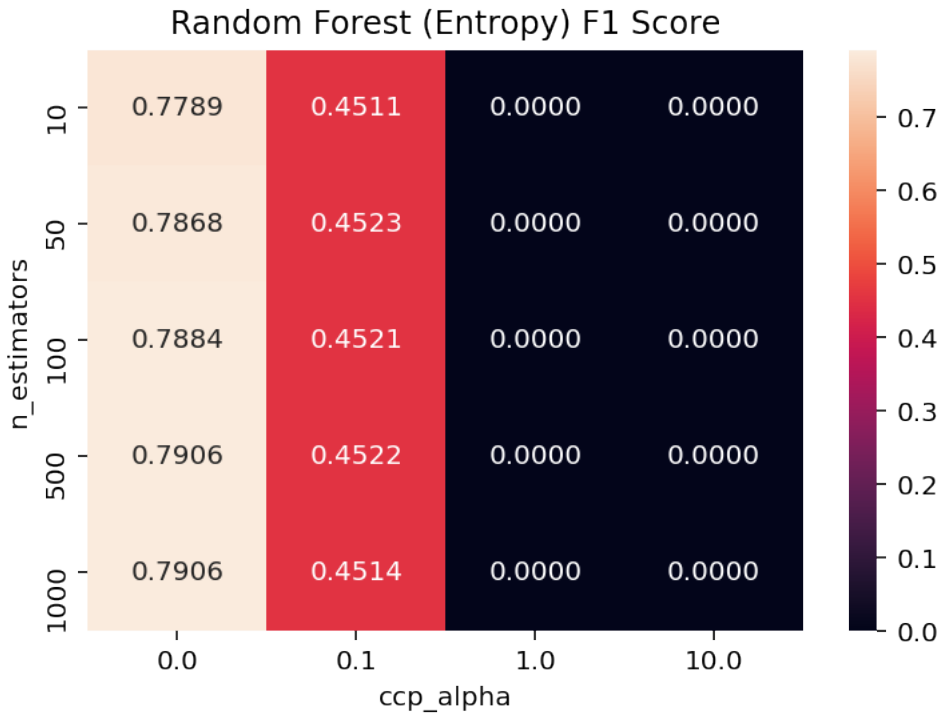
```
sns.heatmap(param_search_results_ranfor_acc_entropy.pivot('n_estimators',␣
 ↪'ccp_alpha', 'score_acc'), annot=True, fmt='.4f')
plt.xlabel('ccp_alpha')
plt.title('Random Forest (Entropy) Accuracy Score')
plt.savefig('results/param_search_results_ranfor_entropy_acc.png')
plt.show()

param_search_results_ranfor_acc_gini =␣
 ↪param_search_results_ranfor_acc[param_search_results_ranfor_acc['criterion']␣
 ↪== 'gini']
param_search_results_ranfor_acc_gini = param_search_results_ranfor_acc_gini.
 ↪drop(columns=['criterion'])

plt.figure()
sns.heatmap(param_search_results_ranfor_acc_gini.pivot('n_estimators',␣
 ↪'ccp_alpha', 'score_acc'), annot=True, fmt='.4f')
plt.xlabel('ccp_alpha')
plt.title('Random Forest (Gini) Accuracy Score')
plt.savefig('results/param_search_results_ranfor_gini_acc.png')
plt.show()
```

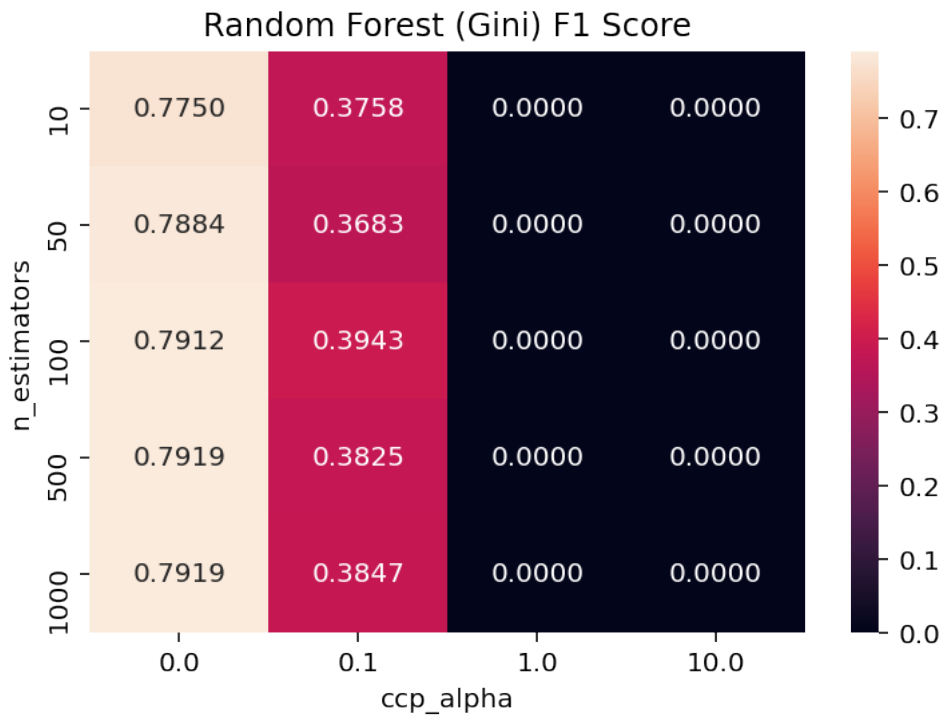




[35]:
```
## F1
param_search_results_ranfor_f1 = pd.DataFrame()

for i in ranfor_f1_idx:
    param_search_results = pd.DataFrame(cv_results[i]['params'])
    param_search_results['score_f1'] = cv_results[i]['mean_test_score']
    param_search_results_ranfor_f1 = pd.concat([param_search_results_ranfor_f1,␣
 ↪param_search_results], ignore_index=True)

param_search_results_ranfor_f1 = param_search_results_ranfor_f1.
 ↪groupby(['n_estimators', 'ccp_alpha', 'criterion'], as_index=True,␣
 ↪group_keys=False).mean()
param_search_results_ranfor_f1.reset_index(inplace=True)

param_search_results_ranfor_f1_entropy =␣
 ↪param_search_results_ranfor_f1[param_search_results_ranfor_f1['criterion']␣
 ↪== 'entropy']
param_search_results_ranfor_f1_entropy = param_search_results_ranfor_f1_entropy.
 ↪drop(columns=['criterion'])

plt.figure()
sns.heatmap(param_search_results_ranfor_f1_entropy.pivot('n_estimators',␣
 ↪'ccp_alpha', 'score_f1'), annot=True, fmt='.4f')
```

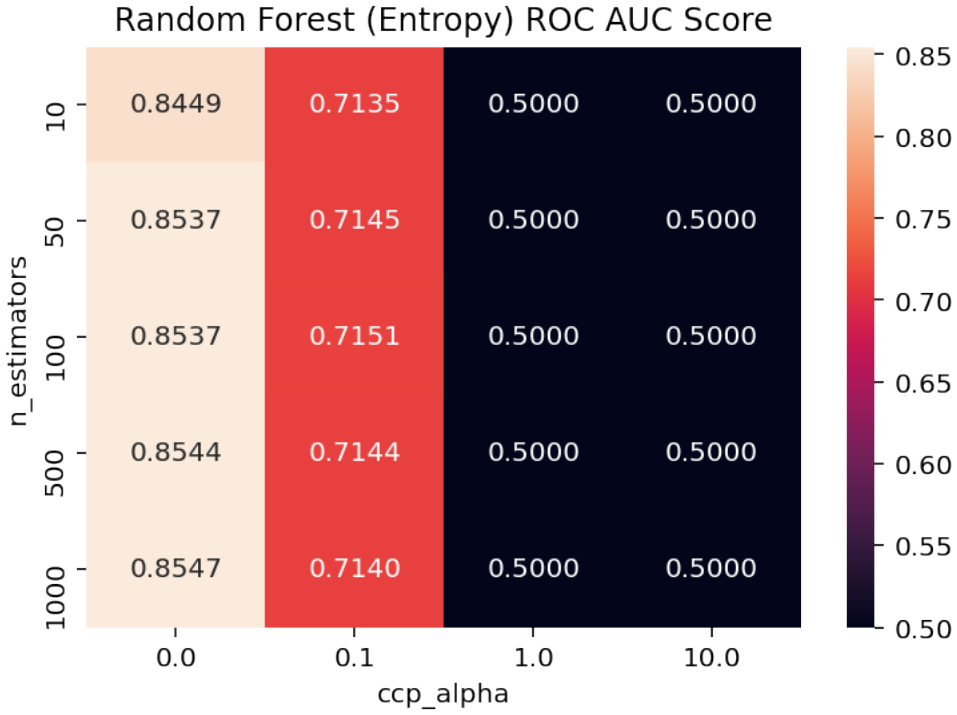
```
plt.xlabel('ccp_alpha')
plt.title('Random Forest (Entropy) F1 Score')
plt.savefig('results/param_search_results_ranfor_entropy_f1.png')
plt.show()

param_search_results_ranfor_f1_gini =␣
↪param_search_results_ranfor_f1[param_search_results_ranfor_f1['criterion']␣
↪== 'gini']
param_search_results_ranfor_f1_gini = param_search_results_ranfor_f1_gini.
↪drop(columns=['criterion'])

plt.figure()
sns.heatmap(param_search_results_ranfor_f1_gini.pivot('n_estimators',␣
↪'ccp_alpha', 'score_f1'), annot=True, fmt='.4f')
plt.xlabel('ccp_alpha')
plt.title('Random Forest (Gini) F1 Score')
plt.savefig('results/param_search_results_ranfor_gini_f1.png')
plt.show()
```

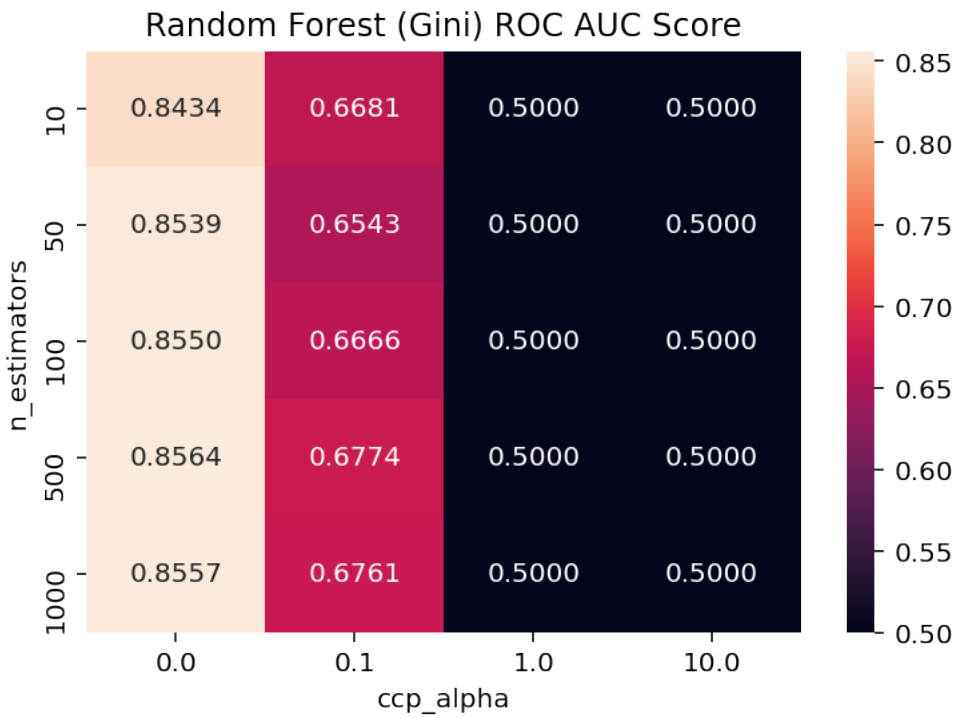




[36]:
```
## ROC AUC
param_search_results_ranfor_rocauc = pd.DataFrame()

for i in ranfor_rocauc_idx:
    param_search_results = pd.DataFrame(cv_results[i]['params'])
    param_search_results['score_rocauc'] = cv_results[i]['mean_test_score']
    param_search_results_ranfor_rocauc = pd.
 ↪concat([param_search_results_ranfor_rocauc, param_search_results],␣
 ↪ignore_index=True)

param_search_results_ranfor_rocauc = param_search_results_ranfor_rocauc.
 ↪groupby(['n_estimators', 'ccp_alpha', 'criterion'], as_index=True,␣
 ↪group_keys=False).mean()
param_search_results_ranfor_rocauc.reset_index(inplace=True)

param_search_results_ranfor_rocauc_entropy =␣
 ↪param_search_results_ranfor_rocauc[param_search_results_ranfor_rocauc['criterion']␣
 ↪== 'entropy']
param_search_results_ranfor_rocauc_entropy =␣
 ↪param_search_results_ranfor_rocauc_entropy.drop(columns=['criterion'])

plt.figure()
```

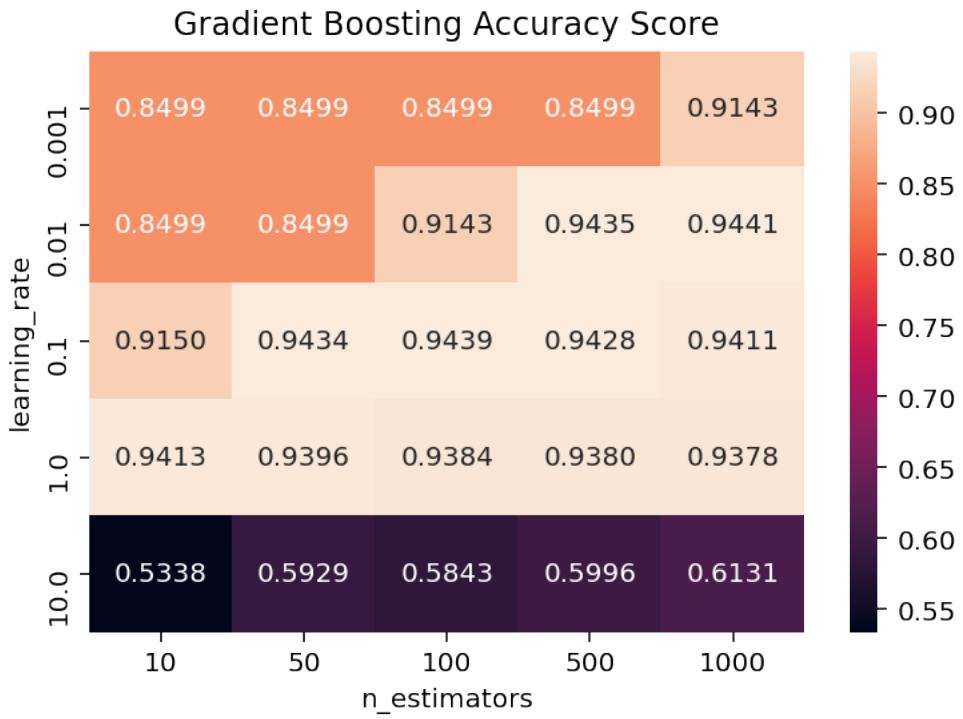
```
sns.heatmap(param_search_results_ranfor_rocauc_entropy.pivot('n_estimators',␣
 ↪'ccp_alpha', 'score_rocauc'), annot=True, fmt='.4f')
plt.xlabel('ccp_alpha')
plt.title('Random Forest (Entropy) ROC AUC Score')
plt.savefig('results/param_search_results_ranfor_entropy_rocauc.png')
plt.show()

param_search_results_ranfor_rocauc_gini =␣
 ↪param_search_results_ranfor_rocauc[param_search_results_ranfor_rocauc['criterion']␣
 ↪== 'gini']
param_search_results_ranfor_rocauc_gini =␣
 ↪param_search_results_ranfor_rocauc_gini.drop(columns=['criterion'])

plt.figure()
sns.heatmap(param_search_results_ranfor_rocauc_gini.pivot('n_estimators',␣
 ↪'ccp_alpha', 'score_rocauc'), annot=True, fmt='.4f')
plt.xlabel('ccp_alpha')
plt.title('Random Forest (Gini) ROC AUC Score')
plt.savefig('results/param_search_results_ranfor_gini_rocauc.png')
plt.show()
```



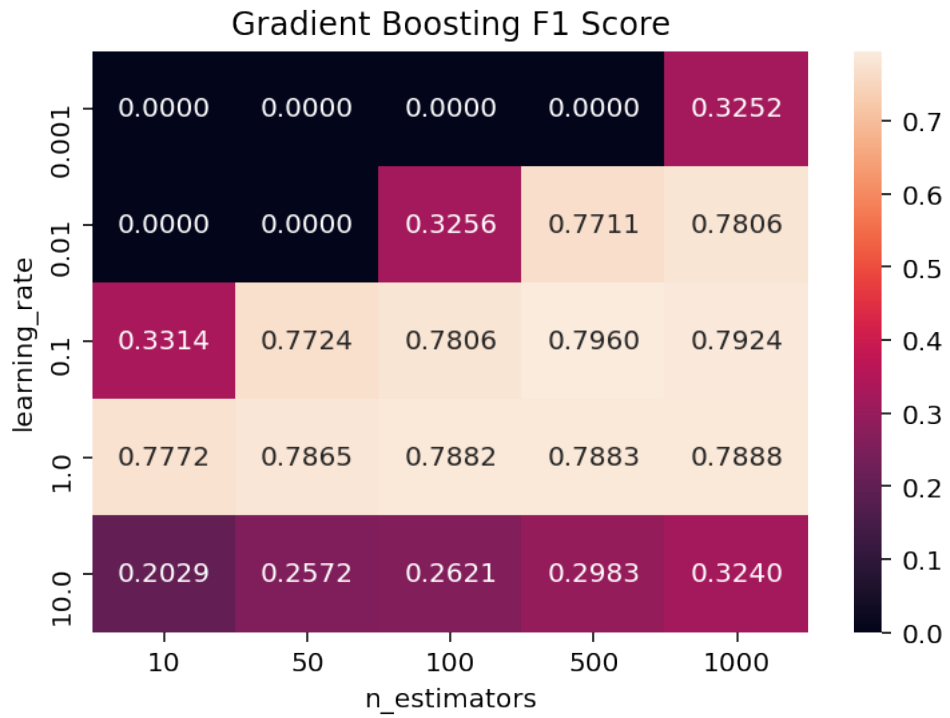


Gradient Boosting Classifier

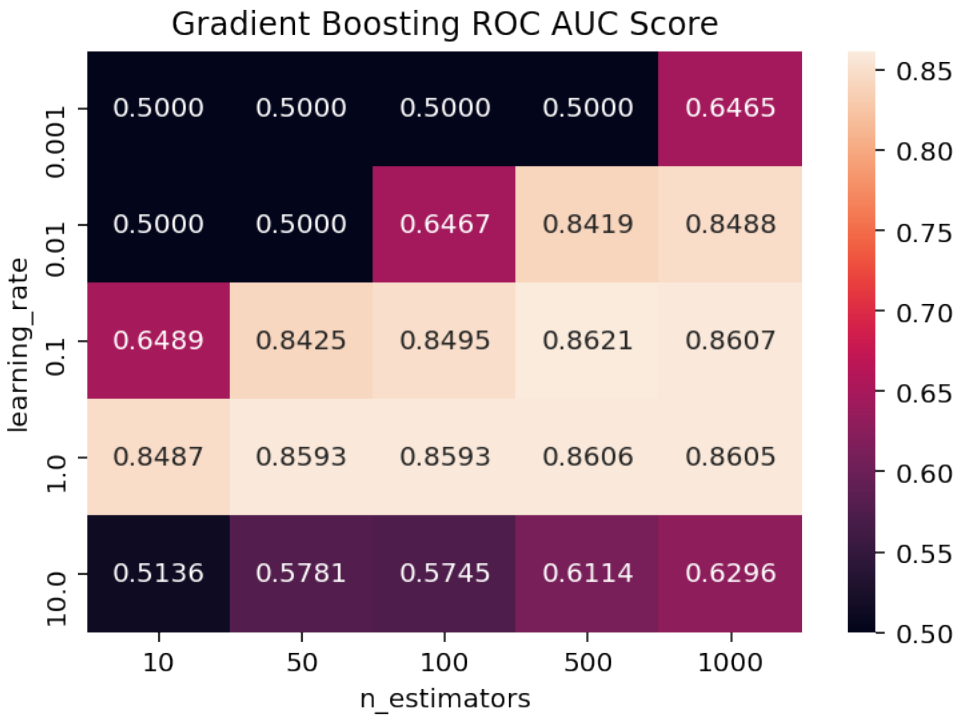
[37]:
```
## Accuracy
param_search_results_graboo_acc = pd.DataFrame()

for i in graboo_acc_idx:
    param_search_results = pd.DataFrame(cv_results[i]['params'])
    param_search_results['score_acc'] = cv_results[i]['mean_test_score']
    param_search_results_graboo_acc = pd.
 ↪concat([param_search_results_graboo_acc, param_search_results],␣
 ↪ignore_index=True)

param_search_results_graboo_acc = param_search_results_graboo_acc.
 ↪groupby(['learning_rate', 'n_estimators'], as_index=True, group_keys=False).
 ↪mean()
param_search_results_graboo_acc.reset_index(inplace=True)
sns.heatmap(param_search_results_graboo_acc.pivot('learning_rate',␣
 ↪'n_estimators', 'score_acc'), annot=True, fmt='.4f')
plt.xlabel('n_estimators')
plt.title('Gradient Boosting Accuracy Score')
plt.savefig('results/param_search_results_graboo_acc.png')
plt.show()
```



[38]:
```
## F1
param_search_results_graboo_f1 = pd.DataFrame()

for i in graboo_f1_idx:
    param_search_results = pd.DataFrame(cv_results[i]['params'])
    param_search_results['score_f1'] = cv_results[i]['mean_test_score']
    param_search_results_graboo_f1 = pd.concat([param_search_results_graboo_f1,␣
 ↪param_search_results], ignore_index=True)

param_search_results_graboo_f1 = param_search_results_graboo_f1.
 ↪groupby(['learning_rate', 'n_estimators'], as_index=True, group_keys=False).
 ↪mean()
param_search_results_graboo_f1.reset_index(inplace=True)
sns.heatmap(param_search_results_graboo_f1.pivot('learning_rate',␣
 ↪'n_estimators', 'score_f1'), annot=True, fmt='.4f')
plt.xlabel('n_estimators')
plt.title('Gradient Boosting F1 Score')
plt.savefig('results/param_search_results_graboo_f1.png')
plt.show()
```



[39]:
```
## ROC AUC
param_search_results_graboo_rocauc = pd.DataFrame()

for i in graboo_rocauc_idx:
    param_search_results = pd.DataFrame(cv_results[i]['params'])
    param_search_results['score_rocauc'] = cv_results[i]['mean_test_score']
    param_search_results_graboo_rocauc = pd.
 ↪concat([param_search_results_graboo_rocauc, param_search_results],␣
 ↪ignore_index=True)

param_search_results_graboo_rocauc = param_search_results_graboo_rocauc.
 ↪groupby(['learning_rate', 'n_estimators'], as_index=True, group_keys=False).
 ↪mean()
param_search_results_graboo_rocauc.reset_index(inplace=True)
sns.heatmap(param_search_results_graboo_rocauc.pivot('learning_rate',␣
 ↪'n_estimators', 'score_rocauc'), annot=True, fmt='.4f')
plt.xlabel('n_estimators')
plt.title('Gradient Boosting ROC AUC Score')
plt.savefig('results/param_search_results_graboo_rocauc.png')
plt.show()
```


Gradient Boosting ROC AUC Score
learning_rate
0.001
0.01
0.1
1.0
10.0
0.5000 0.5000 0.5000 0.5000 0.6465
0.5000 0.5000 0.6467 0.8419 0.8488
0.6489 0.8425 0.8495 0.8621 0.8607
0.8487 0.8593 0.8593 0.8606 0.8605
0.5136 0.5781 0.5745 0.6114 0.6296
10
50
100
500
1000
n_estimators
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50