A Comparison of Binary Classification Learning Algorithms

Aditya Ashar A15519191
Maneesha Nagabandi A15635394

Abstract

This paper's based on the work conducted by Rich Caruana and Alexandru Niculescu-Mizil in their publication, "An Empirical Comparison of Supervised Learning Algorithms," which we will be referring to as CNMO6. We will be utilizing multiple datasets in order to draw a comparison between five supervised learning algorithms: Logistic Regression, Support Vector Machines, K-Nearest Neighbors, Artificial Neural Networks, and Random Forests.

1 INTRODUCTION

Modelled after STATLOG, a study performed on learning algorithms,CNMO6 focuses on empirically comparing ten supervised learning models using eight performance metrics. In our study, we will be conducting binary classification using five of these supervised learning algorithms on a number of datasets. Along with searching for optimal hyperparameters with each of these models, we will be empirically comparing them using three different performance metrics: accuracy, the F1 score, and ROC-AUC.

While CNMO6 performed calibrations and a comparison before and after, we will not be conducting any such comparison. Instead we will be basing our comparison specifically on the calculated performance metrics.

Based on the collected data, it appears that random forests have the best performance both overall and for each measured metric. It will become apparent in later sections, however, that different datasets call for different models in order to optimize performance.

2 METHOD

2.1 Learning algorithms

This section details the hyperparameters we aim to explore and common variations of these parameters that may optimize the algorithm and its performance.

- **Logistic Regression (LR)**: we trained the model, varying the solver between saga and lbfgs; the penalty between L1, L2, and none; and the C value from 10^{-8} and 10^{-4} by factors of 10.
- **K-Nearest Neighbors (KNN)**: we trained the model using n_neighbors values ranging from 1 to 105, incremented by 4.
- **Artificial Neural Networks (ANN)**: we trained the model using hidden_layer_sizes varying between 1, 2, 4, 8, 32, and 128. We also altered the momentum values between 0, 0.2, 0.5, and 0.9.
- **Decision Trees (DT)**: we trained the model using the criterion gini and entropy; the splitters best and random; max depth ranging from 2 to 6; and min_samples_leaf ranging from 1 to 4.
- **Random Forests (RF)**: we trained the model using an n_estimators value of 1024 and max features varying between 1, 2, 4, 6, 8, 12, 16, and 20.

2.2 Datasets

Shown in Table 1, this study uses six datasets from the UCI Machine Learning Repository for the purpose of binary classification.

- SKIN: This dataset contains three attributes in the form of red, green, and blue components and class labels "skin" and "non-skin." 1 was used to represent "skin," and 0 was used to represent "non-skin."
- MUSHROOM: This datasets contains 22 categorical attributes determining class labels of "poisonous" or "nonpoisonous." 1 was used to represent "poisonous," and 0 was used to represent "nonpoisonous."
- ADULT**:** This dataset contains 14 integer and categorical labels determining class labels of "<= 50K" and "> 50K" to denote income predictions. 1 was used to represent "> 50K," and 0 was used to represent "<= 50K."
- LETTER1: This dataset was originally a multiclass classification problem. For the purpose of the study, it was manipulated such that class labels of letters A through M were labeled as 1, and the rest as 0.
- LETTER2: This dataset was derived from the same parent dataset as LETTER1; however, in order to study a highly unbalanced dataset, it was manipulated such that class labels of the letter O were labeled as 0 and the rest as 1.
- BANK: This dataset contains 14 attributes determining a class label of "yes" or "no" denoting a presence of a term deposit subscription. 1 was used to represent "yes," and 0 was used to represent "no."

Table 1: Dataset characteristics

Datasets	Attributes	Training Size	Test Size	% poz
SKIN	3	5000	240,057	79%
MUSHROOM	22/117	5000	3124	48%
ADULT	14/108	5000	43842	24%
LETTER1	16	5000	15000	50%
LETTER2	16	5000	15000	96%
BANK	14/44	5000	40,211	12%

2.3 Performance metrics

The optimal combination of parameters for each of the five models will be found using the following performance metrics: accuracy, F1 score, and ROC_AUC.

Table 2- Comparing performance metrics (averaged over all datasets)

Model	ACC	F1	ROC_AUC	MEAN
LogReg	0.846	0.870*	0.851	0.856
KNN	0.894*	0.920*	0.898*	0.904*
ANN	0.911*	0.933*	0.916	0.920*
DT	0.877*	0.896*	0.878	0.884*
RF	**0.912**	**0.933**	**0.918**	**0.921**

3 EXPERIMENT

For each of the classification problems, we randomly selected 5000 data points to train each model and allotted the rest of the dataset to the test set. We used a five-fold cross validation technique resulting in 5 trials for each problem. The purpose of this study is to select the best combination of parameters for each algorithm that produce the best performance on the test set.

Table 2 contains three performance metrics averaged over the six datasets, used to score each of the algorithms. For each classification problem, we find the optimal combination of parameters by examining the three measured metrics. . Each of the entries in Table 2 average the scores of these five trials across the three metrics. Higher scores correspond to a better performance. The last column, labeled "MEAN," indicates the average calculated over the six problems, the three metrics, and the five trials.

The values indicated in bold show the algorithm with the best performance for the given performance metric. For example, the value bolded in the second column indicates the algorithm that resulted in the highest accuracy score. The values that have a * appearing next to them, have no statistically significant difference to the optimal models. This was determined using t-tests analyzing the un-averaged data from each of the trials for each of the classification problems.

Table 3- Comparing problems by algorithms across all datasets (averaged over three metrics)

Model	SKIN	MSHRM	ADULT	LETT1	LETT2	BANK	Mean
LogReg	0.891	0.999*	0.803	0.758	0.931	0.750	0.855
KNN	0.997*	0.999	0.767	**0.959***	0.993*	0.706	0.904
ANN	**0.998**	0.999	0.797	0.957*	**0.995**	0.772*	0.919*
DT	0.983*	0.999*	0.795	0.821	0.965	0.738	0.884
RF	0.997	**1.000**	**0.807**	0.956	0.992	**0.807**	**0.921**

Table 3 contains a score for each classification problem averaged over the three metrics and the five trails. Each entry represents the averaged performance of the given learning algorithm executed on the given dataset. The last column, labeled "MEAN," indicates the average calculated over all of the datasets; each value in this column shows the averaged performance of the chosen algorithm.

As the table demonstrates, no single algorithm is best for every problem. Based on the calculated mean values, Random Forests appear to have the best overall performance; however, the Artificial Neural Networks actually outperformed on three of the six datasets.

4 DISCUSSION

Based on the performance metrics, it is evident that overall the optimized Random Forest algorithm appears to be the best model. The Random Forest algorithm was found to have the highest accuracy, F1 score, and ROC_AUC values, suggesting that this algorithm really does have the best performance overall. It is important to note that there was variation between each trial, as demonstrated by Appendix table A2. Another important observation is that all of the algorithms performed very similarly in terms of their F1 score, as none of them had a statistically significant difference (Table 2). This suggests that different algorithms do work better in different situations.

This is further emphasized in Table 3, as Random Forests were optimal for four of the six datasets. Artificial neural networks display performance metrics that are mostly insignificantly different from those of Random Forests, once again suggesting that the situation may be driven by the dataset and hyperparameters of choice.

It also appears that the Logistic Regression classifier performed the worst on a majority of the datasets. Similarly, the BANK dataset presented the worst performance metrics when compared to the other datasets.

In summary, we were able to find the optimal hyperparameters for five different learning algorithms using six different datasets. By gathering performance metrics on each of these problems, we were able to perform a comparison on both the overall performance and the situational performance.

5 REFERENCES

1. Caruana, R. and Niculescu-Mizil, A., 2021. An Empirical ComparisonOf Supervised Learning Algorithms. [online] Cs.cornell.edu. Available at: https://www.cs.cornell.edu/~caruana/ctp/ct.papers/caruana.icml06.pdf7
2. Archive.ics.uci.edu. 2021. UCI Ma-chine Learning Repository. [online] Available at: https://archive.ics.uci.edu/ml/index.php [Accessed 1 March 2021].
3. GitHub. 2021. Scikit-Learn/Scikit-Learn. [online] Available at: https://github.com/scikit-learn/scikit-learn/blob/95d4f0841/sklearn/neural_network/_multilayer_perceptron.py [Accessed 1 March 2021].
4. Scikit-learn.org. 2021. Sklearn.Neighbors.Kneighborsclassifier—Scikit-Learn 0.24.1 Documentation. [online] Available at: https://scikit-learn.org/stable/modules/gener

ated/sklearn.neighbors.KNeighborsClassifier.html [Accessed 1 March 2021].
5. Scikit-learn.org. 2021. Sklearn.LinearModel.Logisticregression—Scikit-Learn 0.24.1 Documentation. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html [Accessed 1 March 2021].
6. Scikit-learn.org. 2021. Sklearn.Tree.DecisionTreeClassifier—Scikit-Learn 0.24.1 Documentation. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html [Accessed 1 March 2021].
7. Scikit-learn.org. 2021. Sklearn.Ensemble.RandomForestClassifier—Scikit-Learn 0.24.1 Documentation. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html [Accessed 1 March 2021].

APPENDIX

Table A1: Mean training set performance across all datasets and learning algorithms

Model	SKIN	MSHRM	ADULT	LETT1	LETT2	BANK
LogReg	0.917	0.999	0.858	0.730	0.962	0.899
KNN	1.000	1.000	1.000	1.000	1.000	1.000
ANN	0.998	1.000	0.973	0.999	0.999	0.999
DT	0.991	0.999	0.861	0.810	0.980	0.920
RF	1.000	1.000	1.000	1.000	1.000	1.000

Table A1 displays the mean training set performances for all of the learning algorithms across each of the datasets when using the optimal hyperparameters. Many of the measured training accuracy values are actually very high. While the MUSHROOM dataset has a high training performance, the test set accuracy is actually very similar. On the other hand, the BANK dataset's training accuracy is much higher than the corresponding test accuracy suggesting that the models may have overfit to the training data. This poses a problem as the performance still suffers, suggesting that more regularization must occur when using this particular dataset.

Table A2: Raw test set scores across three metrics for all datasets and algorithms

	SKIN			MUSHROOM			ADULT			LETT1			LETT2			BANK		
	ACC	F1	AUC	ACC	F1	AUC	ACC	F1	AUC	ACC	F1	AUC	ACC	F1	AUC	ACC	F1	AUC

LR	.79	.79	.95	1.0	1.0	1.0	.65	.65	.90	.73	.73	.82	.97	.98	.86	.44	.44	.91
	.82	.82	.96	1.0	1.0	1.0	.65	.65	.90	.72	.72	.81	.98	.98	.85	.47	.47	.90
	.79	.79	.95	1.0	1.0	1.0	.65	.65	.90	.72	.72	.81	.98	.98	.84	.43	.43	.90
	.80	.80	.95	.99	.99	.99	.67	.67	.90	.73	.74	.82	.98	.98	.86	.47	.47	.90
	.82	.82	.95	.99	.99	.99	.66	.66	.91	.73	.73	.82	.98	.98	.84	.45	.45	.90
KNN	.99	.99	.99	1.0	1.0	1.0	.83	.59	.87	.94	.94	.98	.99	.99	.99	.89	.36	.86
	.99	.99	.99	1.0	1.0	1.0	.83	.58	.88	.95	.95	.98	.99	.99	.99	.88	.39	.85
	.99	.99	.99	1.0	1.0	1.0	.83	.58	.87	.95	.95	.98	.99	.99	.99	.88	.37	.85
	.99	.99	.99	.99	.99	.99	.83	.61	.87	.95	.95	.99	.99	.99	.99	.89	.40	.84
	.99	.99	.99	.99	.99	.99	.84	.60	.88	.94	.94	.98	.99	.99	.99	.89	.38	.85
ANN	.99	.99	.99	.99	.99	1.0	.84	.64	.87	.94	.94	.91	.99	.99	.99	.90	.50	.87
	.99	.99	.99	1.0	1.0	1.0	.85	.65	.89	.94	.94	.99	.99	.99	.99	.89	.54	.90
	.99	.99	.99	1.0	1.0	1.0	.84	.65	.89	.95	.95	.99	.99	.99	.99	.89	.51	.90
	.99	.99	.99	.99	.99	1.0	.85	.67	.89	.94	.94	.99	.99	.99	.99	.90	.51	.90
	.99	.99	.99	.99	.99	1.0	.86	.66	.89	.94	.94	.99	.99	.99	.99	.90	.53	.89
DT	.99	.98	.99	1.0	1.0	1.0	.84	.62	.88	.81	.80	.89	.97	.99	.95	.90	.51	.85
	.99	.97	.99	1.0	.99	1.0	.87	.66	.89	.79	.78	.88	.98	.99	.94	.89	.45	.85
	.99	.96	.99	1.0	1.0	1.0	.85	.64	.89	.78	.78	.87	.97	.98	.93	.89	.46	.86
	.99	.97	.99	.99	1.0	1.0	.85	.64	.88	.79	.79	.88	.97	.99	.93	.89	.45	.85
	.99	.97	.99	.99	.99	.99	.86	.66	.90	.80	.79	.89	.97	.99	.94	.90	.48	.85
RF	.99	.99	.99	1.0	1.0	1.0	.84	.65	.90	.94	.94	.99	.99	.99	.99	.91	.52	.92
	.99	.99	.99	1.0	1.0	1.0	.86	.66	.91	.94	.94	.99	.99	.99	.99	.90	.53	.92
	.99	.99	.99	1.0	1.0	1.0	.85	.66	.90	.94	.94	.99	.99	.99	.99	.89	.46	.92
	.99	.99	.99	1.0	1.0	1.0	.86	.68	.90	.94	.94	.99	.99	.99	.99	.90	.49	.92
	.99	.99	.99	1.0	1.0	1.0	.86	.67	.91	.94	.94	.99	.99	.99	.99	.90	.52	.93

Table A3: P-values for Table 2 (RF compared with LOGREG/KNN/ANN/DT)

	Pval-LOGREG	Pval-KNN	Pval-ANN	Pval-DT
RF-ACC	.00057	.81697	.99999	.18121
RF-F1	.19727	.45879	.39457	.54318
RF-ROC_AUC	1.14437	.16783	.00123	.00892
RF_MEAN	.00184	.25118	.18684	.10899

Table A4: P-values for Table 3 (RF versus rest over all datasets)

	RF
SKIN (LOGREG)	1.47042e-05
SKIN (KNN)	.05932
SKIN (ANN)	.00869
SKIN (DT)	4.70269e-06
MEAN (LOGREG)	9.50498e-12
MUSH (LOGREG)	.05532
MUSH (KNN)	.04678
MUSH (ANN)	.02692
MUSH (DT)	.07119
MEAN (KNN)	9.82293e-06
ADULT (LOGREG)	.00901
ADULT (KNN)	1.01813e-05
ADULT (ANN)	1.08454e-06
ADULT (DT)	.00113
MEAN (ANN)	.18684
LETTER1 (LOGREG)	8.02627e-16
LETTER1 (KNN)	.10869

LETTER1 (ANN)	.21119
LETTER1 (DT)	4.04233e-12
MEAN (DT)	2.10808e-10
LETTER2 (LOGREG)	.00201
LETTER2 (KNN)	.14723
LETTER2 (ANN)	1.76546
LETTER2 (DT)	.00078
BANK (LOGREG)	.00198
BANK (KNN)	.00010
BANK (ANN)	.56151
BANK (DT)	.00086

Python Code:

logreg

March 17, 2021

[22]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets
from sklearn import model_selection
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
```

[23]:
```
adult = pd.read_csv("adult.data", header = None)
mushroom = pd.read_csv("mushroom.data", header = None)
bank = pd.read_csv("bank.csv", delimiter = ";")
letterrecog1 = pd.read_csv("letterrecog.data", header = None)
letterrecog = pd.read_csv("letterrecog.data", header = None)
skin = pd.read_csv("skin.txt", delimiter = "\t", header = None)
```

[24]:
```
skin_y = skin[3]
skin_x = skin.drop(3, axis = 1)
skin.head(1000)

bank_y = bank["y"]
bank_y = bank_y.replace({'no': 0, 'yes': 1})
bank_x = bank.drop(["y"], axis = 1)
bank_x = pd.get_dummies(bank_x)
```

```
letterrecog[0] = letterrecog[0].replace(['A', 'B', 'C', 'D', 'E', 'F', 'G',␣
↪'H', 'I', 'J', 'K', 'L', 'M'], 0)
letterrecog[0] = letterrecog[0].replace(['N', 'O', 'P', 'Q', 'R', 'S', 'T',␣
↪'U', 'V', 'W', 'X', 'Y', 'Z'], 1)
letter_y = letterrecog[0]
letter_x = letterrecog.drop([0],axis=1)

letterrecog1[0] = letterrecog1[0].replace(['O'], 0)
letterrecog1[0] = letterrecog1[0].replace(['A', 'B', 'C', 'D', 'E', 'F', 'G',␣
↪'H', 'I', 'J', 'K', 'L', 'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',␣
↪'X', 'Y', 'Z'], 1)
letter1_y = letterrecog1[0]
letter1_x = letterrecog1.drop(0, axis = 1)

adult[14] = adult[14].replace({' <=50K': 0, ' >50K': 1})
adult_y = adult[14]
adult_x = adult.drop([14], axis = 1)
adult_x = pd.get_dummies(adult_x)

mushroom[0] = mushroom[0].replace({'p': 1, 'e': 0})
mush_y = mushroom[0]
mush_x = mushroom.drop(0, axis = 1)
mush_x = pd.get_dummies(mush_x)
```

[25]:
```
metrics_log_mushroom = []
metrics_values_mushroom = []

for x in range(5):
    X = mush_x
    Y = mush_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
↪LogisticRegression())])

    search_space = [{'classifier': [LogisticRegression(max_iter = 10000)],
                     'classifier__solver': ['saga'],
                     'classifier__penalty': ['l1', 'l2'],
                     'classifier__C': np.logspace(-8, 4, 13)},
                     {'classifier': [LogisticRegression(max_iter = 10000)],
                     'classifier__solver': ['lbfgs'],
                     'classifier__penalty': ['l2'],
                     'classifier__C': np.logspace(-8, 4, 13)},
                     {'classifier': [LogisticRegression(max_iter = 10000)],
                     'classifier__solver': ['lbfgs','saga'],
                     'classifier__penalty': ['none']}
                     ]
```

```
    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
↪StratifiedKFold(n_splits=5),
                    scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                    verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_log_mushroom.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_log_mushroom.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_log_mushroom.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_mushroom.append(best_model.
↪cv_results_['mean_test_accuracy'][ np.argmin(best_model.
↪cv_results_['rank_test_accuracy']) ])
    metrics_values_mushroom.append(best_model.cv_results_['mean_test_f1'][ np.
↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_mushroom.append(best_model.cv_results_['mean_test_roc_auc'][␣
↪np.argmin(best_model.cv_results_['rank_test_roc_auc']) ])
print(metrics_values_mushroom)
print(metrics_log_mushroom)
```

```
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.9994, 0.9993820803295572,
0.9999295523774071, 0.9998000000000001, 0.9997901364113325, 0.9999759491083132]
[{'classifier': LogisticRegression(max_iter=10000), 'classifier__C': 10.0,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 10.0,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1.0, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1.0, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1.0, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.1, 'classifier__penalty':
'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1.0, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1.0, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.1, 'classifier__penalty':
'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.01,
'classifier__penalty': 'l2', 'classifier__solver': 'saga'}, {'classifier':
```

```
LogisticRegression(max_iter=10000), 'classifier__C': 0.01,
'classifier__penalty': 'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.01,
'classifier__penalty': 'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1.0, 'classifier__penalty':
'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1.0, 'classifier__penalty':
'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.1, 'classifier__penalty':
'l2', 'classifier__solver': 'saga'}]
```

[26]:
```
metrics_log_letter1 = []
metrics_values_letter1 = []

for x in range(5):
    X = letter1_x
    Y = letter1_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪LogisticRegression())])

    search_space = [{'classifier': [LogisticRegression(max_iter = 10000)],
                     'classifier__solver': ['saga'],
                     'classifier__penalty': ['l1', 'l2'],
                     'classifier__C': np.logspace(-8, 4, 13)},
                     {'classifier': [LogisticRegression(max_iter = 10000)],
                     'classifier__solver': ['lbfgs'],
                     'classifier__penalty': ['l2'],
                     'classifier__C': np.logspace(-8, 4, 13)},
                     {'classifier': [LogisticRegression(max_iter = 10000)],
                     'classifier__solver': ['lbfgs','saga'],
                     'classifier__penalty': ['none']}
                     ]

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
 ↪StratifiedKFold(n_splits=5),
                     scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                     verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_log_letter1.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_log_letter1.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
```

```
    metrics_log_letter1.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_letter1.append(best_model.cv_results_['mean_test_accuracy'][␣
 ↪np.argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_letter1.append(best_model.cv_results_['mean_test_f1'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_letter1.append(best_model.cv_results_['mean_test_roc_auc'][␣
 ↪np.argmin(best_model.cv_results_['rank_test_roc_auc']) ])
print(metrics_values_letter1)
print(metrics_log_letter1)
```

```
[0.9593999999999999, 0.9792793074350721, 0.8580654361864918, 0.9632,
0.9812550514456342, 0.846165243655258, 0.9606, 0.9799040472894921,
0.8451457928940206, 0.9635999999999999, 0.9814625562727153, 0.8642619599278965,
0.9650000000000001, 0.9821882951653944, 0.8424041450777203]
[{'classifier': LogisticRegression(max_iter=10000), 'classifier__C': 1e-08,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1e-08,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.01,
'classifier__penalty': 'l2', 'classifier__solver': 'lbfgs'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1e-08,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1e-08,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.01,
'classifier__penalty': 'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1e-08,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1e-08,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.1, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1e-08,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1e-08,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.01,
'classifier__penalty': 'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1e-08,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1e-08,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.01,
'classifier__penalty': 'l2', 'classifier__solver': 'saga'}]
```

[27]:
```
metrics_log_skin = []
metrics_values_skin = []

for x in range(5):
    X = skin_x
    Y = skin_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪LogisticRegression())])

    search_space = [{'classifier': [LogisticRegression(max_iter = 10000)],
                     'classifier__solver': ['saga'],
                     'classifier__penalty': ['l1', 'l2'],
                     'classifier__C': np.logspace(-8, 4, 13)},
                     {'classifier': [LogisticRegression(max_iter = 10000)],
                     'classifier__solver': ['lbfgs'],
                     'classifier__penalty': ['l2'],
                     'classifier__C': np.logspace(-8, 4, 13)},
                     {'classifier': [LogisticRegression(max_iter = 10000)],
                     'classifier__solver': ['lbfgs','saga'],
                     'classifier__penalty': ['none']}
                     ]

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
 ↪StratifiedKFold(n_splits=5),
                     scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                     verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_log_skin.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_log_skin.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_log_skin.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_skin.append(best_model.cv_results_['mean_test_accuracy'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_skin.append(best_model.cv_results_['mean_test_f1'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_skin.append(best_model.cv_results_['mean_test_roc_auc'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])
print(metrics_values_skin)
print(metrics_log_skin)
```

```
[0.9075999999999999, 0.786988622695977, 0.9472877928306817, 0.9263999999999999,
0.8251045790195587, 0.9565529026394944, 0.9162000000000001, 0.7946894246004156,
0.9487275729152576, 0.9198000000000001, 0.7983778935234352, 0.9520484153527631,
0.9218, 0.817530089865848, 0.9527263798494608]
[{'classifier': LogisticRegression(max_iter=10000), 'classifier__C': 10.0,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 10.0,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__penalty': 'none',
'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 10.0,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 10.0,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1000.0,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 10.0,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 10.0,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 10000.0,
'classifier__penalty': 'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1.0, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1.0, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1000.0,
'classifier__penalty': 'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1.0, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1.0, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 10000.0,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}]
```

[28]:
```
metrics_log_bank = []
metrics_values_bank = []

for x in range(5):
    X = bank_x
    Y = bank_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪LogisticRegression())])

    search_space = [{'classifier': [LogisticRegression(max_iter = 10000)],
```

```
                    'classifier__solver': ['saga'],
                    'classifier__penalty': ['l1', 'l2'],
                    'classifier__C': np.logspace(-8, 4, 13)},
                    {'classifier': [LogisticRegression(max_iter = 10000)],
                    'classifier__solver': ['lbfgs'],
                    'classifier__penalty': ['l2'],
                    'classifier__C': np.logspace(-8, 4, 13)},
                    {'classifier': [LogisticRegression(max_iter = 10000)],
                    'classifier__solver': ['lbfgs','saga'],
                    'classifier__penalty': ['none']}
                    ]

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
↪StratifiedKFold(n_splits=5),
                    scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                    verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_log_bank.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_log_bank.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_log_bank.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_bank.append(best_model.cv_results_['mean_test_accuracy'][ np.
↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_bank.append(best_model.cv_results_['mean_test_f1'][ np.
↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_bank.append(best_model.cv_results_['mean_test_roc_auc'][ np.
↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])
print(metrics_values_bank)
```

```
[0.9022, 0.44217572133729444, 0.9076338923490169, 0.8924, 0.4724843378073258,
0.8974646732067194, 0.8916000000000001, 0.427716898262386, 0.8979985352875193,
0.9022, 0.46673204419425024, 0.8971463161535802, 0.8974, 0.4465196692274261,
0.9077215380221701]
```

[29]:
```
metrics_log_letter = []
metrics_values_letter = []

for x in range(5):
    X = letter_x
    Y = letter_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
↪train_size = 5000, random_state = (10 * x))
```

```
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
↪LogisticRegression())])

    search_space = [{'classifier': [LogisticRegression(max_iter = 10000)],
                     'classifier__solver': ['saga'],
                     'classifier__penalty': ['l1', 'l2'],
                     'classifier__C': np.logspace(-8, 4, 13)},
                     {'classifier': [LogisticRegression(max_iter = 10000)],
                     'classifier__solver': ['lbfgs'],
                     'classifier__penalty': ['l2'],
                     'classifier__C': np.logspace(-8, 4, 13)},
                     {'classifier': [LogisticRegression(max_iter = 10000)],
                     'classifier__solver': ['lbfgs','saga'],
                     'classifier__penalty': ['none']}
                     ]

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
↪StratifiedKFold(n_splits=5),
                     scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                     verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_log_letter.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_log_letter.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_log_letter.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_letter.append(best_model.cv_results_['mean_test_accuracy'][␣
↪np.argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_letter.append(best_model.cv_results_['mean_test_f1'][ np.
↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_letter.append(best_model.cv_results_['mean_test_roc_auc'][␣
↪np.argmin(best_model.cv_results_['rank_test_roc_auc']) ])
print(metrics_values_letter)
print(metrics_log_letter)
```

```
[0.7346, 0.733481022102547, 0.817850551040127, 0.726, 0.7249284357533791,
0.811482000048866, 0.723, 0.7229849503673251, 0.8152916550007534,
0.7305999999999999, 0.7350522116576401, 0.8166769774235945, 0.7322,
0.7298177422174373, 0.8172815717999301]
[{'classifier': LogisticRegression(max_iter=10000), 'classifier__C': 1.0,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1.0, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
```

```
LogisticRegression(max_iter=10000), 'classifier__C': 10.0,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 100.0,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 100.0,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 10000.0,
'classifier__penalty': 'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1.0, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1.0, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 10000.0,
'classifier__penalty': 'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1.0, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1.0, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 10000.0,
'classifier__penalty': 'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 10.0,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 10.0,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__penalty': 'none',
'classifier__solver': 'saga'}]
```

[30]:
```
metrics_log_adult = []
metrics_values_adult = []

for x in range(5):
    X = adult_x
    Y = adult_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪LogisticRegression())])

    search_space = [{'classifier': [LogisticRegression(max_iter = 10000)],
                     'classifier__solver': ['saga'],
                     'classifier__penalty': ['l1', 'l2'],
                     'classifier__C': np.logspace(-8, 4, 13)},
                     {'classifier': [LogisticRegression(max_iter = 10000)],
                     'classifier__solver': ['lbfgs'],
                     'classifier__penalty': ['l2'],
                     'classifier__C': np.logspace(-8, 4, 13)},
                     {'classifier': [LogisticRegression(max_iter = 10000)],
```

```
                    'classifier__solver': ['lbfgs','saga'],
                    'classifier__penalty': ['none']}
                    ]

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
↪StratifiedKFold(n_splits=5),
                    scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                    verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_log_adult.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_log_adult.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_log_adult.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_adult.append(best_model.cv_results_['mean_test_accuracy'][␣
↪np.argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_adult.append(best_model.cv_results_['mean_test_f1'][ np.
↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_adult.append(best_model.cv_results_['mean_test_roc_auc'][ np.
↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])
print(metrics_values_adult)
print(metrics_log_adult)
```

```
[0.8443999999999999, 0.6465204019598264, 0.8993221043193984, 0.8542,
0.6493796243856014, 0.9038224910142307, 0.8482, 0.6471540625387119,
0.9003911031874621, 0.8526, 0.6692540979018624, 0.9027059802301747,
0.8591999999999999, 0.6629474438093611, 0.909544651608934]
[{'classifier': LogisticRegression(max_iter=10000), 'classifier__C': 0.1,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 10.0,
'classifier__penalty': 'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.1, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.1, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 10.0,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.1, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.1, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.1, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
```

```
LogisticRegression(max_iter=10000), 'classifier__C': 0.1, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 10.0,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 10.0,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.1, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1.0, 'classifier__penalty':
'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1.0, 'classifier__penalty':
'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.1, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}]
```

[31]:
```
print(metrics_log_bank)
```

```
[{'classifier': LogisticRegression(max_iter=10000), 'classifier__C': 0.1,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 10.0,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.1, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.1, 'classifier__penalty':
'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1.0, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.1, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.1, 'classifier__penalty':
'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1.0, 'classifier__penalty':
'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.1, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.1, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1.0, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.1, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.1, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 1.0, 'classifier__penalty':
'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=10000), 'classifier__C': 0.1, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}]
```

[34]:
```
metrics_train_mushroom = []
metrics_test_mushroom = []

for x in range(5):
    X = mush_x
    Y = mush_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪LogisticRegression())])

    search_space = [{'classifier': [LogisticRegression(max_iter = 10000)],
                     'classifier__solver': ['saga'],
                     'classifier__penalty': ['l1'],
                     'classifier__C': [0.1]}
                     ]

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
 ↪StratifiedKFold(n_splits=5),
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_mushroom.append(clf.score(X_train, Y_train))
    metrics_test_mushroom.append(clf.score(X_test, Y_test))

print(metrics_train_mushroom)
print(metrics_test_mushroom)
```

```
[0.9996, 0.9994, 0.9994, 0.9996, 0.9998]
[0.9996798975672215, 1.0, 0.9996798975672215, 0.9996798975672215,
0.9990396927016645]
```

[35]:
```
metrics_train_skin = []
metrics_test_skin = []

for x in range(5):
    X = skin_x
    Y = skin_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪LogisticRegression())])

    search_space = [{'classifier': [LogisticRegression(max_iter = 10000)],
                     'classifier__solver': ['saga'],
                     'classifier__penalty': ['l1'],
```

```
                    'classifier__C': [0.1]}
                    ]

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
↪StratifiedKFold(n_splits=5),
                    verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_skin.append(clf.score(X_train, Y_train))
    metrics_test_skin.append(clf.score(X_test, Y_test))

print(metrics_train_skin)
print(metrics_test_skin)
```

```
[0.9066, 0.9246, 0.915, 0.919, 0.9208]
[0.9154617445023474, 0.92021894799985, 0.9177778610913242, 0.9173238022636291,
0.9186693160374411]
```

[36]:
```
metrics_train_adult = []
metrics_test_adult = []

for x in range(5):
    X = adult_x
    Y = adult_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
↪LogisticRegression())])

    search_space = [{'classifier': [LogisticRegression(max_iter = 10000)],
                    'classifier__solver': ['saga'],
                    'classifier__penalty': ['l1'],
                    'classifier__C': [0.1]}
                    ]

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
↪StratifiedKFold(n_splits=5),
                    verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_adult.append(clf.score(X_train, Y_train))
    metrics_test_adult.append(clf.score(X_test, Y_test))

print(metrics_train_adult)
print(metrics_test_adult)
```

```
[0.8506, 0.861, 0.8542, 0.859, 0.8642]
[0.8516019012372555, 0.8469576575595951, 0.8482638510939371, 0.8498240267044012,
0.8493523457058888]
```

[37]:
```
metrics_train_letter = []
metrics_test_letter = []

for x in range(5):
    X = letter_x
    Y = letter_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪LogisticRegression())])

    search_space = [{'classifier': [LogisticRegression(max_iter = 10000)],
                     'classifier__solver': ['saga'],
                     'classifier__penalty': ['l1'],
                     'classifier__C': [0.1]}
                     ]

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
 ↪StratifiedKFold(n_splits=5),
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_letter.append(clf.score(X_train, Y_train))
    metrics_test_letter.append(clf.score(X_test, Y_test))

print(metrics_train_letter)
print(metrics_test_letter)
```

```
[0.734, 0.726, 0.7204, 0.7336, 0.7346]
[0.721, 0.7225333333333334, 0.7235333333333334, 0.7218666666666667, 0.7196]
```

[38]:
```
metrics_train_letter1 = []
metrics_test_letter1 = []

for x in range(5):
    X = letter1_x
    Y = letter1_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪LogisticRegression())])
```

```
    search_space = [{'classifier': [LogisticRegression(max_iter = 10000)],
                     'classifier__solver': ['saga'],
                     'classifier__penalty': ['l1'],
                     'classifier__C': [0.1]}
                     ]

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
↪StratifiedKFold(n_splits=5),
                     verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_letter1.append(clf.score(X_train, Y_train))
    metrics_test_letter1.append(clf.score(X_test, Y_test))

print(metrics_train_letter1)
print(metrics_test_letter1)
```

```
[0.9594, 0.9632, 0.9606, 0.9636, 0.965]
[0.9633333333333334, 0.9620666666666666, 0.9628, 0.9618666666666666,
0.9614666666666667]
```

[39]:
```
metrics_train_bank = []
metrics_test_bank = []

for x in range(5):
    X = bank_x
    Y = bank_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
↪LogisticRegression())])

    search_space = [{'classifier': [LogisticRegression(max_iter = 10000)],
                     'classifier__solver': ['saga'],
                     'classifier__penalty': ['l1'],
                     'classifier__C': [0.1]}
                     ]

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
↪StratifiedKFold(n_splits=5),
                     verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_bank.append(clf.score(X_train, Y_train))
    metrics_test_bank.append(clf.score(X_test, Y_test))
```

```
print(metrics_train_bank)
print(metrics_test_bank)
```

```
[0.9044, 0.8954, 0.8942, 0.9028, 0.8988]
[0.9007485513914103, 0.9024396309467558, 0.9017433040710253, 0.9019671234239387,
0.9012956653651986]
```

knn

March 17, 2021

[28]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets
from sklearn import model_selection
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import KNeighborsClassifier
```

[29]:
```
adult = pd.read_csv("adult.data", header = None)
mushroom = pd.read_csv("mushroom.data", header = None)
bank = pd.read_csv("bank.csv", delimiter = ";")
letterrecog = pd.read_csv("letterrecog.data", header = None)
skin = pd.read_csv("skin.txt", delimiter = "\t", header = None)
letterrecog1 = pd.read_csv("letterrecog.data", header = None)
```

[30]:
```
skin_y = skin[3]
skin_x = skin.drop(3, axis = 1)
skin.head(1000)

bank_y = bank["y"]
bank_y = bank_y.replace({'no': 0, 'yes': 1})
bank_x = bank.drop(["y"], axis = 1)
bank_x = pd.get_dummies(bank_x)
```

```
letterrecog[0] = letterrecog[0].replace(['A', 'B', 'C', 'D', 'E', 'F', 'G',␣
↪'H', 'I', 'J', 'K', 'L', 'M'], 0)
letterrecog[0] = letterrecog[0].replace(['N', 'O', 'P', 'Q', 'R', 'S', 'T',␣
↪'U', 'V', 'W', 'X', 'Y', 'Z'], 1)
letter_y = letterrecog[0]
letter_x = letterrecog.drop([0],axis=1)

letterrecog1[0] = letterrecog1[0].replace(['O'], 0)
letterrecog1[0] = letterrecog1[0].replace(['A', 'B', 'C', 'D', 'E', 'F', 'G',␣
↪'H', 'I', 'J', 'K', 'L', 'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',␣
↪'X', 'Y', 'Z'], 1)
letter1_y = letterrecog1[0]
letter1_x = letterrecog1.drop(0, axis = 1)

adult[14] = adult[14].replace({' <=50K': 0, ' >50K': 1})
adult_y = adult[14]
adult_x = adult.drop([14], axis = 1)
adult_x = pd.get_dummies(adult_x)

mushroom[0] = mushroom[0].replace({'p': 1, 'e': 0})
mush_y = mushroom[0]
mush_x = mushroom.drop(0, axis = 1)
mush_x = pd.get_dummies(mush_x)
```

[22]:
```
metrics_knn_skin = []
metrics_values_skin = []

for x in range(5):
    X = skin_x
    Y = skin_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
↪train_size = 5000, random_state = (10 * x))
    pipe_knn = Pipeline([('std', StandardScaler()), ('classifier',␣
↪KNeighborsClassifier())])

    neighbors = range(1, 106, 4)
    search_space = {'classifier__n_neighbors': neighbors}

    clf = GridSearchCV(pipe_knn, search_space, cv = StratifiedKFold(n_splits=5),
                       scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_knn_skin.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
```

```
    metrics_knn_skin.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_knn_skin.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_skin.append(best_model.cv_results_['mean_test_accuracy'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_skin.append(best_model.cv_results_['mean_test_f1'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_skin.append(best_model.cv_results_['mean_test_roc_auc'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])
print(metrics_values_skin)
print(metrics_knn_skin)
```

```
[0.9984, 0.9962746621603266, 0.9993962930484217, 0.9986, 0.9966054508474391,
0.9996669150756784, 0.9976, 0.9940715631932495, 0.9992869587566673, 0.9972,
0.9928701185087376, 0.9992221691352127, 0.9976, 0.9943060418097286,
0.9997857273583002]
[{'classifier__n_neighbors': 1}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 25}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 1}, {'classifier__n_neighbors': 17},
{'classifier__n_neighbors': 1}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 9}, {'classifier__n_neighbors': 5},
{'classifier__n_neighbors': 5}, {'classifier__n_neighbors': 5},
{'classifier__n_neighbors': 1}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 33}]
```

[23]:
```
metrics_knn_bank = []
metrics_values_bank = []

for x in range(5):
    X = bank_x
    Y = bank_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_knn = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪KNeighborsClassifier())])

    neighbors = range(1, 106, 4)
    search_space = {'classifier__n_neighbors': neighbors}

    clf = GridSearchCV(pipe_knn, search_space, cv = StratifiedKFold(n_splits=5),
                       scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)
```

```
    metrics_knn_bank.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_knn_bank.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_knn_bank.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_bank.append(best_model.cv_results_['mean_test_accuracy'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_bank.append(best_model.cv_results_['mean_test_f1'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_bank.append(best_model.cv_results_['mean_test_roc_auc'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])
print(metrics_values_bank)
print(metrics_knn_bank)
```

```
[0.8904, 0.36384401503290537, 0.8569548458093218, 0.8808, 0.3917163936203515,
0.8553697767492773, 0.8831999999999999, 0.36831698586081646, 0.8462858414216982,
0.8934, 0.40028555845748925, 0.8421278680963911, 0.8886, 0.381534064409389,
0.8513226181493809]
[{'classifier__n_neighbors': 37}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 101}, {'classifier__n_neighbors': 13},
{'classifier__n_neighbors': 5}, {'classifier__n_neighbors': 101},
{'classifier__n_neighbors': 49}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 105}, {'classifier__n_neighbors': 17},
{'classifier__n_neighbors': 1}, {'classifier__n_neighbors': 101},
{'classifier__n_neighbors': 9}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 101}]
```

[32]:
```
metrics_knn_letter = []
metrics_values_letter = []

for x in range(5):
    X = letter_x
    Y = letter_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_knn = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪KNeighborsClassifier())])

    neighbors = range(1, 106, 4)
    search_space = {'classifier__n_neighbors': neighbors}

    clf = GridSearchCV(pipe_knn, search_space, cv = StratifiedKFold(n_splits=5),
                       scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                       verbose = 0)
```

```
    best_model = clf.fit(X_train, Y_train)

    metrics_knn_letter.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_knn_letter.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_knn_letter.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_letter.append(best_model.cv_results_['mean_test_accuracy'][␣
 ↪np.argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_letter.append(best_model.cv_results_['mean_test_f1'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_letter.append(best_model.cv_results_['mean_test_roc_auc'][␣
 ↪np.argmin(best_model.cv_results_['rank_test_roc_auc']) ])

print(metrics_values_letter)
print(metrics_knn_letter)
```

```
[0.9443999999999999, 0.9448772908157576, 0.9834827179290511, 0.9458,
0.9457846476329509, 0.9834278938565377, 0.9501999999999999, 0.9504897097403239,
0.9846602027714292, 0.9479999999999998, 0.9492630553423774, 0.9855071338903029,
0.9448000000000001, 0.9448245378332271, 0.9812550319610389]
[{'classifier__n_neighbors': 1}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 5}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 1}, {'classifier__n_neighbors': 5},
{'classifier__n_neighbors': 1}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 5}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 1}, {'classifier__n_neighbors': 5},
{'classifier__n_neighbors': 1}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 5}]
```

[25]:
```
metrics_knn_letter1 = []
metrics_values_letter1 = []

for x in range(5):
    X = letter1_x
    Y = letter1_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_knn = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪KNeighborsClassifier())])

    neighbors = range(1, 106, 4)
    search_space = {'classifier__n_neighbors': neighbors}
```

```
    clf = GridSearchCV(pipe_knn, search_space, cv = StratifiedKFold(n_splits=5),
                       scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_knn_letter1.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_knn_letter1.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_knn_letter1.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_letter1.append(best_model.cv_results_['mean_test_accuracy'][␣
 ↪np.argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_letter1.append(best_model.cv_results_['mean_test_f1'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_letter1.append(best_model.cv_results_['mean_test_roc_auc'][␣
 ↪np.argmin(best_model.cv_results_['rank_test_roc_auc']) ])
print(metrics_values_letter1)
print(metrics_knn_letter1)
```

```
[0.9904, 0.9949859552330796, 0.9945953570474835, 0.9902, 0.9949064810363015,
0.9928141250083993, 0.9904, 0.9949978062967325, 0.993187745345727, 0.9896,
0.9945963156421588, 0.9945767560032985, 0.9906, 0.9951233371257567,
0.9932760917838639]
[{'classifier__n_neighbors': 1}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 5}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 1}, {'classifier__n_neighbors': 13},
{'classifier__n_neighbors': 1}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 13}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 1}, {'classifier__n_neighbors': 9},
{'classifier__n_neighbors': 1}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 13}]
```

[26]:
```
metrics_knn_mush = []
metrics_values_mush = []

for x in range(5):
    X = mush_x
    Y = mush_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_knn = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪KNeighborsClassifier())])
```

```
    neighbors = range(1, 106, 4)
    search_space = {'classifier__n_neighbors': neighbors}

    clf = GridSearchCV(pipe_knn, search_space, cv = StratifiedKFold(n_splits=5),
                       scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_knn_mush.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_knn_mush.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_knn_mush.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_mush.append(best_model.cv_results_['mean_test_accuracy'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_mush.append(best_model.cv_results_['mean_test_f1'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_mush.append(best_model.cv_results_['mean_test_roc_auc'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])
print(metrics_values_mush)
print(metrics_knn_mush)
```

```
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.9994, 0.9993820803295572,
0.9998326766429775, 0.9998000000000001, 0.9997901364113325, 0.9999931849764208]
[{'classifier__n_neighbors': 1}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 1}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 1}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 1}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 1}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 1}, {'classifier__n_neighbors': 37},
{'classifier__n_neighbors': 1}, {'classifier__n_neighbors': 1},
{'classifier__n_neighbors': 13}]
```

[33]:
```
metrics_knn_adult = []
metrics_values_adult = []

for x in range(5):
    X = adult_x
    Y = adult_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_knn = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪KNeighborsClassifier())])
```

```
    neighbors = range(1, 106, 4)
    search_space = {'classifier__n_neighbors': neighbors}

    clf = GridSearchCV(pipe_knn, search_space, cv = StratifiedKFold(n_splits=5),
                       scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_knn_adult.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_knn_adult.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_knn_adult.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_adult.append(best_model.cv_results_['mean_test_accuracy'][␣
 ↪np.argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_adult.append(best_model.cv_results_['mean_test_f1'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_adult.append(best_model.cv_results_['mean_test_roc_auc'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])
print(metrics_values_adult)
print(metrics_knn_adult)
```

```
[0.826, 0.5974798034133271, 0.8733100443365643, 0.833, 0.578699020319533,
0.8784279370870373, 0.827, 0.5778683664585215, 0.8677767069695934, 0.8336,
0.612577561553796, 0.8743990826673282, 0.8421999999999998, 0.603276595593697,
0.8769401736417425]
[{'classifier__n_neighbors': 85}, {'classifier__n_neighbors': 17},
{'classifier__n_neighbors': 105}, {'classifier__n_neighbors': 101},
{'classifier__n_neighbors': 13}, {'classifier__n_neighbors': 105},
{'classifier__n_neighbors': 73}, {'classifier__n_neighbors': 13},
{'classifier__n_neighbors': 105}, {'classifier__n_neighbors': 85},
{'classifier__n_neighbors': 25}, {'classifier__n_neighbors': 105},
{'classifier__n_neighbors': 53}, {'classifier__n_neighbors': 21},
{'classifier__n_neighbors': 81}]
```

[34]:
```
metrics_train_skin = []
metrics_test_skin = []

for x in range(5):
    X = skin_x
    Y = skin_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
```

```
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
↪KNeighborsClassifier())])

    search_space = {'classifier__n_neighbors': [1]}

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
↪StratifiedKFold(n_splits=5),
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_skin.append(clf.score(X_train, Y_train))
    metrics_test_skin.append(clf.score(X_test, Y_test))

print(metrics_train_skin)
print(metrics_test_skin)
```

```
[1.0, 1.0, 1.0, 1.0, 1.0]
[0.998271243912904, 0.9985545099705486, 0.9986336578395965, 0.9978546761810737,
0.9984337053283179]
```

[35]:
```
metrics_train_mush = []
metrics_test_mush = []

for x in range(5):
    X = mush_x
    Y = mush_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
↪KNeighborsClassifier())])

    search_space = {'classifier__n_neighbors': [1]}

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
↪StratifiedKFold(n_splits=5),
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_mush.append(clf.score(X_train, Y_train))
    metrics_test_mush.append(clf.score(X_test, Y_test))

print(metrics_train_mush)
print(metrics_test_mush)
```

```
[1.0, 1.0, 1.0, 1.0, 1.0]
```

```
[1.0, 1.0, 1.0, 1.0, 1.0]
```

[36]:
```
metrics_train_letter = []
metrics_test_letter = []

for x in range(5):
    X = letter_x
    Y = letter_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪KNeighborsClassifier())])

    search_space = {'classifier__n_neighbors': [1]}

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
 ↪StratifiedKFold(n_splits=5),
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_letter.append(clf.score(X_train, Y_train))
    metrics_test_letter.append(clf.score(X_test, Y_test))

print(metrics_train_letter)
print(metrics_test_letter)
```

```
[1.0, 1.0, 1.0, 1.0, 1.0]
[0.9506666666666667, 0.9532, 0.9516, 0.9562666666666667, 0.9505333333333333]
```

[37]:
```
metrics_train_letter1 = []
metrics_test_letter1 = []

for x in range(5):
    X = letter1_x
    Y = letter1_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪KNeighborsClassifier())])

    search_space = {'classifier__n_neighbors': [1]}

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
 ↪StratifiedKFold(n_splits=5),
                       verbose = 0)
```

```
    best_model = clf.fit(X_train, Y_train)

    metrics_train_letter1.append(clf.score(X_train, Y_train))
    metrics_test_letter1.append(clf.score(X_test, Y_test))

print(metrics_train_letter1)
print(metrics_test_letter1)
```

```
[1.0, 1.0, 1.0, 1.0, 1.0]
[0.9914666666666667, 0.9904, 0.9903333333333333, 0.9903333333333333,
0.9887333333333334]
```

[38]:
```
metrics_train_bank = []
metrics_test_bank = []

for x in range(5):
    X = bank_x
    Y = bank_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪KNeighborsClassifier())])

    search_space = {'classifier__n_neighbors': [1]}

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
 ↪StratifiedKFold(n_splits=5),
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_bank.append(clf.score(X_train, Y_train))
    metrics_test_bank.append(clf.score(X_test, Y_test))

print(metrics_train_bank)
print(metrics_test_bank)
```

```
[1.0, 1.0, 1.0, 1.0, 1.0]
[0.8679465817811046, 0.8617791151674915, 0.8641167839645868, 0.866180895774788,
0.8633458506378852]
```

[39]:
```
metrics_train_adult = []
metrics_test_adult = []

for x in range(5):
    X = adult_x
    Y = adult_y
```

```
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
↪KNeighborsClassifier())])

    search_space = {'classifier__n_neighbors': [1]}

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
↪StratifiedKFold(n_splits=5),
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_adult.append(clf.score(X_train, Y_train))
    metrics_test_adult.append(clf.score(X_test, Y_test))

print(metrics_train_adult)
print(metrics_test_adult)
```

```
[1.0, 1.0, 1.0, 1.0, 1.0]
[0.781103733536519, 0.7863285076738871, 0.7705816189543195, 0.7805594862305432,
0.7862922245201553]
```

[ ]:

ann

March 17, 2021

[2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets
from sklearn import model_selection
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
```

[3]:
```
adult = pd.read_csv("adult.data", header = None)
mushroom = pd.read_csv("mushroom.data", header = None)
bank = pd.read_csv("bank.csv", delimiter = ";")
letterrecog1 = pd.read_csv("letterrecog.data", header = None)
letterrecog = pd.read_csv("letterrecog.data", header = None)
skin = pd.read_csv("skin.txt", delimiter = "\t", header = None)
```

[4]:
```
skin_y = skin[3]
skin_x = skin.drop(3, axis = 1)
skin.head(1000)

bank_y = bank["y"]
bank_y = bank_y.replace({'no': 0, 'yes': 1})
bank_x = bank.drop(["y"], axis = 1)
```

```
bank_x = pd.get_dummies(bank_x)

letterrecog[0] = letterrecog[0].replace(['A', 'B', 'C', 'D', 'E', 'F', 'G',␣
↪'H', 'I', 'J', 'K', 'L', 'M'], 0)
letterrecog[0] = letterrecog[0].replace(['N', 'O', 'P', 'Q', 'R', 'S', 'T',␣
↪'U', 'V', 'W', 'X', 'Y', 'Z'], 1)
letter_y = letterrecog[0]
letter_x = letterrecog.drop([0],axis=1)

letterrecog1[0] = letterrecog1[0].replace(['O'], 0)
letterrecog1[0] = letterrecog1[0].replace(['A', 'B', 'C', 'D', 'E', 'F', 'G',␣
↪'H', 'I', 'J', 'K', 'L', 'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',␣
↪'X', 'Y', 'Z'], 1)
letter1_y = letterrecog1[0]
letter1_x = letterrecog1.drop(0, axis = 1)

adult[14] = adult[14].replace({' <=50K': 0, ' >50K': 1})
adult_y = adult[14]
adult_x = adult.drop([14], axis = 1)
adult_x = pd.get_dummies(adult_x)

mushroom[0] = mushroom[0].replace({'p': 1, 'e': 0})
mush_y = mushroom[0]
mush_x = mushroom.drop(0, axis = 1)
mush_x = pd.get_dummies(mush_x)
```

[14]:
```
metrics_ann_skin = []
metrics_values_skin = []

for x in range(5):
    X = skin_x
    Y = skin_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
↪train_size = 5000, random_state = (10 * x))
    pipe_ann = Pipeline([('std', StandardScaler()), ('classifier',␣
↪MLPClassifier())])

    hidden_units = [1, 2, 4, 8, 32, 128]
    momentums = [0, .2, .5, .9]
    search_space = {'classifier__hidden_layer_sizes': hidden_units,␣
↪'classifier__max_iter': [10000], 'classifier__momentum': momentums}

    clf = GridSearchCV(pipe_ann, search_space, cv = StratifiedKFold(n_splits=5),
                       scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                       verbose = 0)
```

```
    best_model = clf.fit(X_train, Y_train)

    metrics_ann_skin.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_ann_skin.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_ann_skin.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_skin.append(best_model.cv_results_['mean_test_accuracy'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_skin.append(best_model.cv_results_['mean_test_f1'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_skin.append(best_model.cv_results_['mean_test_roc_auc'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])
    print("done")
print(metrics_values_skin)
print(metrics_ann_skin)
```

```
done
done
done
done
done
[0.9978, 0.9948824905581454, 0.999923641808496, 0.9986, 0.9966054451277699,
0.9999006910529529, 0.9966000000000002, 0.9916170894593563, 0.9997489398876847,
0.9982, 0.9954094482134043, 0.9995973881191272, 0.9982000000000001,
0.9957424347074116, 0.9999686809734742]
[{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
'classifier__momentum': 0.9}, {'classifier__hidden_layer_sizes': 128,
'classifier__max_iter': 10000, 'classifier__momentum': 0.9},
{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
'classifier__momentum': 0.5}, {'classifier__hidden_layer_sizes': 128,
'classifier__max_iter': 10000, 'classifier__momentum': 0},
{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
'classifier__momentum': 0}, {'classifier__hidden_layer_sizes': 32,
'classifier__max_iter': 10000, 'classifier__momentum': 0.9},
{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
'classifier__momentum': 0.5}, {'classifier__hidden_layer_sizes': 128,
'classifier__max_iter': 10000, 'classifier__momentum': 0.5},
{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
'classifier__momentum': 0}, {'classifier__hidden_layer_sizes': 128,
'classifier__max_iter': 10000, 'classifier__momentum': 0},
{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
'classifier__momentum': 0}, {'classifier__hidden_layer_sizes': 32,
'classifier__max_iter': 10000, 'classifier__momentum': 0.9},
{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
```

```
'classifier__momentum': 0.2}, {'classifier__hidden_layer_sizes': 128,
'classifier__max_iter': 10000, 'classifier__momentum': 0.2},
{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
'classifier__momentum': 0.2}]
```

[16]:
```
metrics_ann_bank = []
metrics_values_bank = []

for x in range(5):
    X = bank_x
    Y = bank_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_ann = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪MLPClassifier())])

    hidden_units = [1, 2, 4, 8, 32, 128]
    momentums = [0, .2, .5, .9]
    search_space = {'classifier__hidden_layer_sizes': hidden_units,␣
 ↪'classifier__max_iter': [10000], 'classifier__momentum': momentums}

    clf = GridSearchCV(pipe_ann, search_space, cv = StratifiedKFold(n_splits=5),
                       scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_ann_bank.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_ann_bank.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_ann_bank.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_bank.append(best_model.cv_results_['mean_test_accuracy'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_bank.append(best_model.cv_results_['mean_test_f1'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_bank.append(best_model.cv_results_['mean_test_roc_auc'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])
    print("done")
print(metrics_values_bank)
print(metrics_ann_bank)
```

```
done
done
done
```

```
done
done
[0.8988000000000002, 0.5038451747129866, 0.9021220811290803, 0.8914,
0.54058287345667, 0.8968738464234173, 0.8922000000000001, 0.5120690728958696,
0.9027597778828416, 0.8996000000000001, 0.5098754906106151, 0.8916224681963181,
0.8991999999999999, 0.5284074402802411, 0.9056856336793313]
[{'classifier__hidden_layer_sizes': 2, 'classifier__max_iter': 10000,
'classifier__momentum': 0.9}, {'classifier__hidden_layer_sizes': 2,
'classifier__max_iter': 10000, 'classifier__momentum': 0.9},
{'classifier__hidden_layer_sizes': 2, 'classifier__max_iter': 10000,
'classifier__momentum': 0.9}, {'classifier__hidden_layer_sizes': 2,
'classifier__max_iter': 10000, 'classifier__momentum': 0.5},
{'classifier__hidden_layer_sizes': 4, 'classifier__max_iter': 10000,
'classifier__momentum': 0}, {'classifier__hidden_layer_sizes': 4,
'classifier__max_iter': 10000, 'classifier__momentum': 0},
{'classifier__hidden_layer_sizes': 4, 'classifier__max_iter': 10000,
'classifier__momentum': 0.2}, {'classifier__hidden_layer_sizes': 4,
'classifier__max_iter': 10000, 'classifier__momentum': 0.2},
{'classifier__hidden_layer_sizes': 4, 'classifier__max_iter': 10000,
'classifier__momentum': 0}, {'classifier__hidden_layer_sizes': 4,
'classifier__max_iter': 10000, 'classifier__momentum': 0.2},
{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
'classifier__momentum': 0.5}, {'classifier__hidden_layer_sizes': 4,
'classifier__max_iter': 10000, 'classifier__momentum': 0.2},
{'classifier__hidden_layer_sizes': 4, 'classifier__max_iter': 10000,
'classifier__momentum': 0.9}, {'classifier__hidden_layer_sizes': 4,
'classifier__max_iter': 10000, 'classifier__momentum': 0.9},
{'classifier__hidden_layer_sizes': 4, 'classifier__max_iter': 10000,
'classifier__momentum': 0}]
```

[17]:
```
metrics_ann_letter = []
metrics_values_letter = []

for x in range(5):
    X = letter_x
    Y = letter_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_ann = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪MLPClassifier())])

    hidden_units = [1, 2, 4, 8, 32, 128]
    momentums = [0, .2, .5, .9]
    search_space = {'classifier__hidden_layer_sizes': hidden_units,␣
 ↪'classifier__max_iter': [10000], 'classifier__momentum': momentums}

    clf = GridSearchCV(pipe_ann, search_space, cv = StratifiedKFold(n_splits=5),
```

```
                    scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                    verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_ann_letter.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_ann_letter.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_ann_letter.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_letter.append(best_model.cv_results_['mean_test_accuracy'][␣
 ↪np.argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_letter.append(best_model.cv_results_['mean_test_f1'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_letter.append(best_model.cv_results_['mean_test_roc_auc'][␣
 ↪np.argmin(best_model.cv_results_['rank_test_roc_auc']) ])
    print("done")
print(metrics_values_letter)
print(metrics_ann_letter)
```

```
done
done
done
done
done
[0.9389999999999998, 0.9393091457494618, 0.9853412535377963, 0.9416,
0.9414246260822754, 0.9873919198744044, 0.9474, 0.9478592912575454,
0.9880109204756881, 0.9428000000000001, 0.9437818614638485, 0.9874993186539631,
0.9414, 0.9414850281626282, 0.9861750824241332]
[{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
'classifier__momentum': 0}, {'classifier__hidden_layer_sizes': 128,
'classifier__max_iter': 10000, 'classifier__momentum': 0},
{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
'classifier__momentum': 0}, {'classifier__hidden_layer_sizes': 128,
'classifier__max_iter': 10000, 'classifier__momentum': 0.9},
{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
'classifier__momentum': 0.9}, {'classifier__hidden_layer_sizes': 128,
'classifier__max_iter': 10000, 'classifier__momentum': 0.5},
{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
'classifier__momentum': 0.2}, {'classifier__hidden_layer_sizes': 128,
'classifier__max_iter': 10000, 'classifier__momentum': 0.2},
{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
'classifier__momentum': 0.2}, {'classifier__hidden_layer_sizes': 128,
'classifier__max_iter': 10000, 'classifier__momentum': 0.2},
{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
```

```
'classifier__momentum': 0.2}, {'classifier__hidden_layer_sizes': 128,
'classifier__max_iter': 10000, 'classifier__momentum': 0.2},
{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
'classifier__momentum': 0}, {'classifier__hidden_layer_sizes': 128,
'classifier__max_iter': 10000, 'classifier__momentum': 0},
{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
'classifier__momentum': 0.2}]
```

[18]:
```
metrics_ann_letter1 = []
metrics_values_letter1 = []

for x in range(5):
    X = letter1_x
    Y = letter1_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_ann = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪MLPClassifier())])

    hidden_units = [1, 2, 4, 8, 32, 128]
    momentums = [0, .2, .5, .9]
    search_space = {'classifier__hidden_layer_sizes': hidden_units,␣
 ↪'classifier__max_iter': [10000], 'classifier__momentum': momentums}

    clf = GridSearchCV(pipe_ann, search_space, cv = StratifiedKFold(n_splits=5),
                       scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_ann_letter1.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_ann_letter1.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_ann_letter1.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_letter1.append(best_model.cv_results_['mean_test_accuracy'][␣
 ↪np.argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_letter1.append(best_model.cv_results_['mean_test_f1'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_letter1.append(best_model.cv_results_['mean_test_roc_auc'][␣
 ↪np.argmin(best_model.cv_results_['rank_test_roc_auc']) ])
    print("done")
print(metrics_values_letter1)
print(metrics_ann_letter1)
```

```
done
done
done
done
done
[0.9944000000000001, 0.9970831128699027, 0.9982729384436702, 0.993,
0.996374718940962, 0.9972474025194387, 0.9902000000000001, 0.9949045327936945,
0.9976562316563943, 0.992, 0.9958563062766084, 0.9965002895050012, 0.993,
0.9963793249155086, 0.9965891931902295]
[{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
'classifier__momentum': 0}, {'classifier__hidden_layer_sizes': 128,
'classifier__max_iter': 10000, 'classifier__momentum': 0},
{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
'classifier__momentum': 0}, {'classifier__hidden_layer_sizes': 32,
'classifier__max_iter': 10000, 'classifier__momentum': 0},
{'classifier__hidden_layer_sizes': 32, 'classifier__max_iter': 10000,
'classifier__momentum': 0}, {'classifier__hidden_layer_sizes': 128,
'classifier__max_iter': 10000, 'classifier__momentum': 0.2},
{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
'classifier__momentum': 0}, {'classifier__hidden_layer_sizes': 128,
'classifier__max_iter': 10000, 'classifier__momentum': 0},
{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
'classifier__momentum': 0.2}, {'classifier__hidden_layer_sizes': 32,
'classifier__max_iter': 10000, 'classifier__momentum': 0},
{'classifier__hidden_layer_sizes': 32, 'classifier__max_iter': 10000,
'classifier__momentum': 0}, {'classifier__hidden_layer_sizes': 128,
'classifier__max_iter': 10000, 'classifier__momentum': 0.9},
{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
'classifier__momentum': 0}, {'classifier__hidden_layer_sizes': 128,
'classifier__max_iter': 10000, 'classifier__momentum': 0},
{'classifier__hidden_layer_sizes': 128, 'classifier__max_iter': 10000,
'classifier__momentum': 0.2}]
```

[19]:
```
metrics_ann_adult = []
metrics_values_adult = []

for x in range(5):
    X = adult_x
    Y = adult_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_ann = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪MLPClassifier())])

    hidden_units = [1, 2, 4, 8, 32, 128]
    momentums = [0, .2, .5, .9]
```

```
    search_space = {'classifier__hidden_layer_sizes': hidden_units,␣
↪'classifier__max_iter': [10000], 'classifier__momentum': momentums}

    clf = GridSearchCV(pipe_ann, search_space, cv = StratifiedKFold(n_splits=5),
                       scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_ann_adult.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_ann_adult.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_ann_adult.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_adult.append(best_model.cv_results_['mean_test_accuracy'][␣
↪np.argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_adult.append(best_model.cv_results_['mean_test_f1'][ np.
↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_adult.append(best_model.cv_results_['mean_test_roc_auc'][ np.
↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])
    print("done")
print(metrics_values_adult)
print(metrics_ann_adult)
```

```
done
done
done
done
done
[0.8412, 0.6431564320208706, 0.8865650916525893, 0.8505999999999998,
0.6501685706776256, 0.8890645991749528, 0.8413999999999999, 0.6495924689453618,
0.8860342180864638, 0.8516, 0.6654727666025264, 0.8928723260287585, 0.8562,
0.6604701264823458, 0.8928082374838995]
[{'classifier__hidden_layer_sizes': 2, 'classifier__max_iter': 10000,
'classifier__momentum': 0.5}, {'classifier__hidden_layer_sizes': 2,
'classifier__max_iter': 10000, 'classifier__momentum': 0.2},
{'classifier__hidden_layer_sizes': 2, 'classifier__max_iter': 10000,
'classifier__momentum': 0.5}, {'classifier__hidden_layer_sizes': 2,
'classifier__max_iter': 10000, 'classifier__momentum': 0},
{'classifier__hidden_layer_sizes': 2, 'classifier__max_iter': 10000,
'classifier__momentum': 0}, {'classifier__hidden_layer_sizes': 2,
'classifier__max_iter': 10000, 'classifier__momentum': 0},
{'classifier__hidden_layer_sizes': 2, 'classifier__max_iter': 10000,
'classifier__momentum': 0.2}, {'classifier__hidden_layer_sizes': 2,
'classifier__max_iter': 10000, 'classifier__momentum': 0.9},
```

```
{'classifier__hidden_layer_sizes': 2, 'classifier__max_iter': 10000,
'classifier__momentum': 0.5}, {'classifier__hidden_layer_sizes': 1,
'classifier__max_iter': 10000, 'classifier__momentum': 0.2},
{'classifier__hidden_layer_sizes': 1, 'classifier__max_iter': 10000,
'classifier__momentum': 0.2}, {'classifier__hidden_layer_sizes': 2,
'classifier__max_iter': 10000, 'classifier__momentum': 0.9},
{'classifier__hidden_layer_sizes': 2, 'classifier__max_iter': 10000,
'classifier__momentum': 0.5}, {'classifier__hidden_layer_sizes': 2,
'classifier__max_iter': 10000, 'classifier__momentum': 0.9},
{'classifier__hidden_layer_sizes': 2, 'classifier__max_iter': 10000,
'classifier__momentum': 0.5}]
```

[20]:
```
metrics_ann_mush = []
metrics_values_mush = []

for x in range(5):
    X = mush_x
    Y = mush_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_ann = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪MLPClassifier())])

    hidden_units = [1, 2, 4, 8, 32, 128]
    momentums = [0, .2, .5, .9]
    search_space = {'classifier__hidden_layer_sizes': hidden_units,␣
 ↪'classifier__max_iter': [10000], 'classifier__momentum': momentums}

    clf = GridSearchCV(pipe_ann, search_space, cv = StratifiedKFold(n_splits=5),
                       scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_ann_mush.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_ann_mush.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_ann_mush.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_mush.append(best_model.cv_results_['mean_test_accuracy'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_mush.append(best_model.cv_results_['mean_test_f1'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_mush.append(best_model.cv_results_['mean_test_roc_auc'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])
```

```
    print("done")
print(metrics_values_mush)
print(metrics_ann_mush)
```

```
done
done
done
done
done
[0.9998000000000001, 0.9997931747673215, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
0.9994, 0.9993820803295572, 1.0, 0.9998000000000001, 0.9997901364113325, 1.0]
[{'classifier__hidden_layer_sizes': 32, 'classifier__max_iter': 10000,
'classifier__momentum': 0}, {'classifier__hidden_layer_sizes': 32,
'classifier__max_iter': 10000, 'classifier__momentum': 0.2},
{'classifier__hidden_layer_sizes': 8, 'classifier__max_iter': 10000,
'classifier__momentum': 0.5}, {'classifier__hidden_layer_sizes': 8,
'classifier__max_iter': 10000, 'classifier__momentum': 0.5},
{'classifier__hidden_layer_sizes': 8, 'classifier__max_iter': 10000,
'classifier__momentum': 0.5}, {'classifier__hidden_layer_sizes': 8,
'classifier__max_iter': 10000, 'classifier__momentum': 0},
{'classifier__hidden_layer_sizes': 32, 'classifier__max_iter': 10000,
'classifier__momentum': 0.9}, {'classifier__hidden_layer_sizes': 32,
'classifier__max_iter': 10000, 'classifier__momentum': 0.9},
{'classifier__hidden_layer_sizes': 32, 'classifier__max_iter': 10000,
'classifier__momentum': 0.2}, {'classifier__hidden_layer_sizes': 8,
'classifier__max_iter': 10000, 'classifier__momentum': 0},
{'classifier__hidden_layer_sizes': 8, 'classifier__max_iter': 10000,
'classifier__momentum': 0}, {'classifier__hidden_layer_sizes': 8,
'classifier__max_iter': 10000, 'classifier__momentum': 0.2},
{'classifier__hidden_layer_sizes': 8, 'classifier__max_iter': 10000,
'classifier__momentum': 0.9}, {'classifier__hidden_layer_sizes': 8,
'classifier__max_iter': 10000, 'classifier__momentum': 0.9},
{'classifier__hidden_layer_sizes': 4, 'classifier__max_iter': 10000,
'classifier__momentum': 0}]
```

[22]:
```
metrics_train_skin = []
metrics_test_skin = []

for x in range(5):
    X = skin_x
    Y = skin_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪MLPClassifier())])
```

```
    search_space = {'classifier__hidden_layer_sizes': [128],␣
↪'classifier__max_iter': [10000], 'classifier__momentum': [0]}

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
↪StratifiedKFold(n_splits=5),
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_skin.append(clf.score(X_train, Y_train))
    metrics_test_skin.append(clf.score(X_test, Y_test))

print(metrics_train_skin)
print(metrics_test_skin)
```

```
[0.9986, 0.9986, 0.9982, 0.9984, 0.999]
[0.9979629837913495, 0.9979046643088934, 0.9986253264849598, 0.9976380609605219,
0.9981171138521268]
```

[23]:
```
metrics_train_mush = []
metrics_test_mush = []

for x in range(5):
    X = mush_x
    Y = mush_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
↪MLPClassifier())])

    search_space = {'classifier__hidden_layer_sizes': [128],␣
↪'classifier__max_iter': [10000], 'classifier__momentum': [0]}

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
↪StratifiedKFold(n_splits=5),
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_mush.append(clf.score(X_train, Y_train))
    metrics_test_mush.append(clf.score(X_test, Y_test))

print(metrics_train_mush)
print(metrics_test_mush)
```

```
[1.0, 1.0, 1.0, 1.0, 1.0]
[1.0, 1.0, 1.0, 0.9996798975672215, 0.9996798975672215]
```

[5]:
```
metrics_train_letter = []
metrics_test_letter = []

for x in range(5):
    X = letter_x
    Y = letter_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
↪MLPClassifier())])

    search_space = {'classifier__hidden_layer_sizes': [128],␣
↪'classifier__max_iter': [10000], 'classifier__momentum': [0]}

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
↪StratifiedKFold(n_splits=5),
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_letter.append(clf.score(X_train, Y_train))
    metrics_test_letter.append(clf.score(X_test, Y_test))

print(metrics_train_letter)
print(metrics_test_letter)
```

```
[0.9998, 1.0, 1.0, 1.0, 1.0]
[0.9430666666666667, 0.9506, 0.9448666666666666, 0.9458, 0.9479333333333333]
```

[6]:
```
metrics_train_letter1 = []
metrics_test_letter1 = []

for x in range(5):
    X = letter1_x
    Y = letter1_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
↪MLPClassifier())])

    search_space = {'classifier__hidden_layer_sizes': [128],␣
↪'classifier__max_iter': [10000], 'classifier__momentum': [0]}

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
↪StratifiedKFold(n_splits=5),
                       verbose = 0)
```

```
    best_model = clf.fit(X_train, Y_train)

    metrics_train_letter1.append(clf.score(X_train, Y_train))
    metrics_test_letter1.append(clf.score(X_test, Y_test))

print(metrics_train_letter1)
print(metrics_test_letter1)
```

```
[1.0, 1.0, 0.9998, 1.0, 1.0]
[0.9921333333333333, 0.9931333333333333, 0.9938, 0.9938, 0.9922]
```

[7]:
```
metrics_train_adult = []
metrics_test_adult = []

for x in range(5):
    X = adult_x
    Y = adult_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪MLPClassifier())])

    search_space = {'classifier__hidden_layer_sizes': [128],␣
 ↪'classifier__max_iter': [10000], 'classifier__momentum': [0]}

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
 ↪StratifiedKFold(n_splits=5),
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_adult.append(clf.score(X_train, Y_train))
    metrics_test_adult.append(clf.score(X_test, Y_test))

print(metrics_train_adult)
print(metrics_test_adult)
```

```
[0.9756, 0.976, 0.9734, 0.969, 0.9724]
[0.8241355538623417, 0.8254417473966837, 0.8131780414353615, 0.8211966184100722,
0.8264213925474402]
```

[8]:
```
metrics_train_bank = []
metrics_test_bank = []

for x in range(5):
    X = bank_x
    Y = bank_y
```

```
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
↪MLPClassifier())])

    search_space = {'classifier__hidden_layer_sizes': [128],␣
↪'classifier__max_iter': [10000], 'classifier__momentum': [0]}

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
↪StratifiedKFold(n_splits=5),
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_bank.append(clf.score(X_train, Y_train))
    metrics_test_bank.append(clf.score(X_test, Y_test))

print(metrics_train_bank)
print(metrics_test_bank)
```

```
[0.9986, 0.9992, 0.9996, 0.9992, 0.9998]
[0.8897316654646739, 0.885578573027281, 0.8826440526224167, 0.8833652483151376,
0.8849568526025217]
```

[ ]:

decisiontree

March 17, 2021

[5]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import datasets
from sklearn import model_selection
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
```

[6]:
```
adult = pd.read_csv("adult.data", header = None)
mushroom = pd.read_csv("mushroom.data", header = None)
bank = pd.read_csv("bank.csv", delimiter = ";")
letterrecog1 = pd.read_csv("letterrecog.data", header = None)
letterrecog = pd.read_csv("letterrecog.data", header = None)
skin = pd.read_csv("skin.txt", delimiter = "\t", header = None)
```

[7]:
```
skin_y = skin[3]
skin_x = skin.drop(3, axis = 1)
skin.head(1000)

bank_y = bank["y"]
bank_y = bank_y.replace({'no': 0, 'yes': 1})
bank_x = bank.drop(["y"], axis = 1)
bank_x = pd.get_dummies(bank_x)
```

```
letterrecog[0] = letterrecog[0].replace(['A', 'B', 'C', 'D', 'E', 'F', 'G',␣
↪'H', 'I', 'J', 'K', 'L', 'M'], 0)
letterrecog[0] = letterrecog[0].replace(['N', 'O', 'P', 'Q', 'R', 'S', 'T',␣
↪'U', 'V', 'W', 'X', 'Y', 'Z'], 1)
letter_y = letterrecog[0]
letter_x = letterrecog.drop([0],axis=1)

letterrecog1[0] = letterrecog1[0].replace(['O'], 0)
letterrecog1[0] = letterrecog1[0].replace(['A', 'B', 'C', 'D', 'E', 'F', 'G',␣
↪'H', 'I', 'J', 'K', 'L', 'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',␣
↪'X', 'Y', 'Z'], 1)
letter1_y = letterrecog1[0]
letter1_x = letterrecog1.drop(0, axis = 1)

adult[14] = adult[14].replace({' <=50K': 0, ' >50K': 1})
adult_y = adult[14]
adult_x = adult.drop([14], axis = 1)
adult_x = pd.get_dummies(adult_x)

mushroom[0] = mushroom[0].replace({'p': 1, 'e': 0})
mush_y = mushroom[0]
mush_x = mushroom.drop(0, axis = 1)
mush_x = pd.get_dummies(mush_x)
```

[8]:
```
metrics_svc_skin = []
metrics_values_skin = []

for x in range(5):
    X = skin_x
    Y = skin_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
↪train_size = 5000, random_state = (10 * x))
    pipe_svc = Pipeline([('std', StandardScaler()), ('classifier',␣
↪DecisionTreeClassifier())])

    criterions = ['gini', 'entropy']
    splitters = ['best', 'random']
    max_depths = [2, 3, 4, 5, 6]
    min_samples_leafs = [1, 2, 3, 4]

    search_space = {'classifier__criterion': criterions, 'classifier__splitter':
↪ splitters, 'classifier__max_depth': max_depths,␣
↪'classifier__min_samples_leaf' :                    min_samples_leafs}

    clf = GridSearchCV(pipe_svc, search_space, cv = StratifiedKFold(n_splits=5),
                       scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
```

```
                    verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_svc_skin.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_svc_skin.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_svc_skin.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_skin.append(best_model.cv_results_['mean_test_accuracy'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_skin.append(best_model.cv_results_['mean_test_f1'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_skin.append(best_model.cv_results_['mean_test_roc_auc'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])
    print("done")
print(metrics_values_skin)
print(metrics_svc_skin)
```

```
done
done
done
done
done
[0.9906, 0.9783551223596095, 0.9929714670914092, 0.9872, 0.9694808015693172,
0.991552542390532, 0.985, 0.9636832403125675, 0.9894727555958157, 0.9874,
0.9685258257897829, 0.9926803631151457, 0.9864, 0.9683371724990139,
0.9916108204492871]
[{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 3, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
```

```
'classifier__min_samples_leaf': 2, 'classifier__splitter': 'random'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 2, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 5,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'}]
```

[9]:
```
metrics_svc_mush = []
metrics_values_mush = []

for x in range(5):
    X = mush_x
    Y = mush_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_svc = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪DecisionTreeClassifier())])

    criterions = ['gini', 'entropy']
    splitters = ['best', 'random']
    max_depths = [2, 3, 4, 5, 6]
    min_samples_leafs = [1, 2, 3, 4]

    search_space = {'classifier__criterion': criterions, 'classifier__splitter':
 ↪ splitters, 'classifier__max_depth': max_depths,␣
 ↪'classifier__min_samples_leaf' :                    min_samples_leafs}

    clf = GridSearchCV(pipe_svc, search_space, cv = StratifiedKFold(n_splits=5),
                       scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_svc_mush.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_svc_mush.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_svc_mush.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])
```

```
    metrics_values_mush.append(best_model.cv_results_['mean_test_accuracy'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_mush.append(best_model.cv_results_['mean_test_f1'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_mush.append(best_model.cv_results_['mean_test_roc_auc'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])
    print("done")
print(metrics_values_mush)
print(metrics_svc_mush)
```

```
done
done
done
done
done
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.9994, 0.9993820803295572, 1.0,
0.9998000000000001, 0.9997901364113325, 0.9999995991518051]
[{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 2, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 2, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 2, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 3, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 3, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 3, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 5,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'random'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 5,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'random'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 3, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 5,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 5,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 3, 'classifier__splitter': 'best'}]
```

[10]:
```
metrics_svc_bank = []
metrics_values_bank = []

for x in range(5):
    X = bank_x
    Y = bank_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_svc = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪DecisionTreeClassifier())])

    criterions = ['gini', 'entropy']
    splitters = ['best', 'random']
    max_depths = [2, 3, 4, 5, 6]
    min_samples_leafs = [1, 2, 3, 4]

    search_space = {'classifier__criterion': criterions, 'classifier__splitter':
 ↪ splitters, 'classifier__max_depth': max_depths,␣
 ↪'classifier__min_samples_leaf' :                    min_samples_leafs}

    clf = GridSearchCV(pipe_svc, search_space, cv = StratifiedKFold(n_splits=5),
                    scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                    verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_svc_bank.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_svc_bank.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_svc_bank.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_bank.append(best_model.cv_results_['mean_test_accuracy'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_bank.append(best_model.cv_results_['mean_test_f1'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_bank.append(best_model.cv_results_['mean_test_roc_auc'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])
    print("done")
print(metrics_values_bank)
print(metrics_svc_bank)
```

```
done
done
done
done
```

```
done
[0.9018, 0.5127376773777481, 0.8514674480383858, 0.8855999999999999,
0.4446828338113097, 0.847997518145152, 0.893, 0.4604738371483531,
0.8561809939650138, 0.8948, 0.44491646197578083, 0.8477093662323686,
0.8996000000000001, 0.47864127151463504, 0.8540245104434039]
[{'classifier__criterion': 'entropy', 'classifier__max_depth': 3,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 2,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 4,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 3,
'classifier__min_samples_leaf': 3, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 3,
'classifier__min_samples_leaf': 3, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 4,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'random'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 5,
'classifier__min_samples_leaf': 2, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'random'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 4,
'classifier__min_samples_leaf': 3, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 5,
'classifier__min_samples_leaf': 2, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 3,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 3,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 4,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'}]
```

[11]:
```
metrics_svc_letter = []
metrics_values_letter = []

for x in range(5):
    X = letter_x
    Y = letter_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_svc = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪DecisionTreeClassifier())])
```

```
    criterions = ['gini', 'entropy']
    splitters = ['best', 'random']
    max_depths = [2, 3, 4, 5, 6]
    min_samples_leafs = [1, 2, 3, 4]

    search_space = {'classifier__criterion': criterions, 'classifier__splitter':
↪ splitters, 'classifier__max_depth': max_depths,␣
↪'classifier__min_samples_leaf' :                    min_samples_leafs}

    clf = GridSearchCV(pipe_svc, search_space, cv = StratifiedKFold(n_splits=5),
                    scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                    verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_svc_letter.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_svc_letter.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_svc_letter.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_letter.append(best_model.cv_results_['mean_test_accuracy'][␣
↪np.argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_letter.append(best_model.cv_results_['mean_test_f1'][ np.
↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_letter.append(best_model.cv_results_['mean_test_roc_auc'][␣
↪np.argmin(best_model.cv_results_['rank_test_roc_auc']) ])
    print("done")
print(metrics_values_letter)
print(metrics_svc_letter)
```

```
done
done
done
done
done
[0.8074, 0.7951950913749362, 0.8927800956794988, 0.7882000000000001,
0.7801722213775234, 0.8809747849609169, 0.7796, 0.7775377192671089,
0.8730566744318441, 0.7914, 0.7914780767487232, 0.8810694100842348, 0.798,
0.7918010640923623, 0.8928823012944813]
[{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
```

```
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 2, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 3, 'classifier__splitter': 'best'}]
```

[12]:
```
metrics_svc_letter1 = []
metrics_values_letter1 = []

for x in range(5):
    X = letter1_x
    Y = letter1_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_svc = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪DecisionTreeClassifier())])

    criterions = ['gini', 'entropy']
    splitters = ['best', 'random']
    max_depths = [2, 3, 4, 5, 6]
    min_samples_leafs = [1, 2, 3, 4]

    search_space = {'classifier__criterion': criterions, 'classifier__splitter':
 ↪ splitters, 'classifier__max_depth': max_depths,␣
 ↪'classifier__min_samples_leaf' :                    min_samples_leafs}

    clf = GridSearchCV(pipe_svc, search_space, cv = StratifiedKFold(n_splits=5),
                       scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
```

```
                    verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_svc_letter1.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_svc_letter1.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_svc_letter1.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_letter1.append(best_model.cv_results_['mean_test_accuracy'][␣
 ↪np.argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_letter1.append(best_model.cv_results_['mean_test_f1'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_letter1.append(best_model.cv_results_['mean_test_roc_auc'][␣
 ↪np.argmin(best_model.cv_results_['rank_test_roc_auc']) ])
    print("done")
print(metrics_values_letter1)
print(metrics_svc_letter1)
```

```
done
done
done
done
done
[0.9734, 0.9861674963117689, 0.9459189106650092, 0.9777999999999999,
0.9885446069855736, 0.9350418735620096, 0.9677999999999999, 0.983248389737501,
0.9309169668347608, 0.9718, 0.9853404525761512, 0.9267867697553683, 0.9736,
0.9863523614671139, 0.9432301998519614]
[{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 3, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 3, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 3, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 3, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 2, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
```

```
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 3, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 3, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'random'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 1, 'classifier__splitter': 'best'}]
```

[13]:

```
metrics_svc_adult = []
metrics_values_adult = []

for x in range(5):
    X = adult_x
    Y = adult_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_svc = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪DecisionTreeClassifier())])

    criterions = ['gini', 'entropy']
    splitters = ['best', 'random']
    max_depths = [2, 3, 4, 5, 6]
    min_samples_leafs = [1, 2, 3, 4]

    search_space = {'classifier__criterion': criterions, 'classifier__splitter':
 ↪ splitters, 'classifier__max_depth': max_depths,␣
 ↪'classifier__min_samples_leaf' :                    min_samples_leafs}

    clf = GridSearchCV(pipe_svc, search_space, cv = StratifiedKFold(n_splits=5),
                       scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_svc_adult.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_svc_adult.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_svc_adult.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])
```

```
    metrics_values_adult.append(best_model.cv_results_['mean_test_accuracy'][␣
 ↪np.argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_adult.append(best_model.cv_results_['mean_test_f1'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_adult.append(best_model.cv_results_['mean_test_roc_auc'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])
    print("done")
print(metrics_values_adult)
print(metrics_svc_adult)
```

```
done
done
done
done
done
[0.8406, 0.6190663946269261, 0.8820078832707374, 0.8652, 0.6624313301171425,
0.8945836262491371, 0.8452, 0.6421086844109974, 0.8879380735839792, 0.8446,
0.6410817899601222, 0.8796378332289819, 0.8629999999999999, 0.6562711112586372,
0.8970064564357567]
[{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 4,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'entropy', 'classifier__max_depth': 5,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
{'classifier__criterion': 'gini', 'classifier__max_depth': 5,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'},
```

```
{'classifier__criterion': 'entropy', 'classifier__max_depth': 6,
'classifier__min_samples_leaf': 4, 'classifier__splitter': 'best'}]
```

[15]:
```
metrics_train_skin = []
metrics_test_skin = []

for x in range(5):
    X = skin_x
    Y = skin_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_svc = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪DecisionTreeClassifier())])

    criterions = ['gini']
    splitters = ['best']
    max_depths = [6]
    min_samples_leafs = [1]

    search_space = {'classifier__criterion': criterions, 'classifier__splitter':
 ↪ splitters, 'classifier__max_depth': max_depths,␣
 ↪'classifier__min_samples_leaf' :                    min_samples_leafs}

    clf = GridSearchCV(pipe_svc, search_space, cv = StratifiedKFold(n_splits=5),
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_skin.append(clf.score(X_train, Y_train))
    metrics_test_skin.append(clf.score(X_test, Y_test))

print(metrics_train_skin)
print(metrics_test_skin)
```

```
[0.9914, 0.9914, 0.9894, 0.9916, 0.9904]
[0.9860116555651366, 0.9879778552593759, 0.9875571218502273, 0.9866031817443357,
0.98762377268732]
```

[16]:
```
metrics_train_bank = []
metrics_test_bank = []

for x in range(5):
    X = bank_x
    Y = bank_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
```

```
    pipe_svc = Pipeline([('std', StandardScaler()), ('classifier',␣
↪DecisionTreeClassifier())])

    criterions = ['gini']
    splitters = ['best']
    max_depths = [6]
    min_samples_leafs = [1]

    search_space = {'classifier__criterion': criterions, 'classifier__splitter':
↪ splitters, 'classifier__max_depth': max_depths,␣
↪'classifier__min_samples_leaf' :                    min_samples_leafs}

    clf = GridSearchCV(pipe_svc, search_space, cv = StratifiedKFold(n_splits=5),
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_bank.append(clf.score(X_train, Y_train))
    metrics_test_bank.append(clf.score(X_test, Y_test))

print(metrics_train_bank)
print(metrics_test_bank)
```

```
[0.9274, 0.9136, 0.916, 0.9198, 0.9224]
[0.8922185471637114, 0.8962721643331427, 0.894506478326826, 0.888612568700107,
0.891994727810798]
```

[17]:
```
metrics_train_letter = []
metrics_test_letter = []

for x in range(5):
    X = letter_x
    Y = letter_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
↪train_size = 5000, random_state = (10 * x))
    pipe_svc = Pipeline([('std', StandardScaler()), ('classifier',␣
↪DecisionTreeClassifier())])

    criterions = ['gini']
    splitters = ['best']
    max_depths = [6]
    min_samples_leafs = [1]

    search_space = {'classifier__criterion': criterions, 'classifier__splitter':
↪ splitters, 'classifier__max_depth': max_depths,␣
↪'classifier__min_samples_leaf' :                    min_samples_leafs}
```

```
    clf = GridSearchCV(pipe_svc, search_space, cv = StratifiedKFold(n_splits=5),
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_letter.append(clf.score(X_train, Y_train))
    metrics_test_letter.append(clf.score(X_test, Y_test))

print(metrics_train_letter)
print(metrics_test_letter)
```

```
[0.8052, 0.7986, 0.8236, 0.8112, 0.816]
[0.7856666666666666, 0.7764666666666666, 0.804, 0.8024666666666667,
0.7908666666666667]
```

[18]:
```
metrics_train_letter1 = []
metrics_test_letter1 = []

for x in range(5):
    X = letter1_x
    Y = letter1_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_svc = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪DecisionTreeClassifier())])

    criterions = ['gini']
    splitters = ['best']
    max_depths = [6]
    min_samples_leafs = [1]

    search_space = {'classifier__criterion': criterions, 'classifier__splitter':
 ↪ splitters, 'classifier__max_depth': max_depths,␣
 ↪'classifier__min_samples_leaf' :                    min_samples_leafs}

    clf = GridSearchCV(pipe_svc, search_space, cv = StratifiedKFold(n_splits=5),
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_letter1.append(clf.score(X_train, Y_train))
    metrics_test_letter1.append(clf.score(X_test, Y_test))

print(metrics_train_letter1)
print(metrics_test_letter1)
```

```
[0.978, 0.9852, 0.9786, 0.9828, 0.9744]
```

```
[0.9778, 0.9796666666666667, 0.9785333333333334, 0.9758, 0.9658666666666667]
```

[19]:
```
metrics_train_adult = []
metrics_test_adult = []

for x in range(5):
    X = adult_x
    Y = adult_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_svc = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪DecisionTreeClassifier())])

    criterions = ['gini']
    splitters = ['best']
    max_depths = [6]
    min_samples_leafs = [1]

    search_space = {'classifier__criterion': criterions, 'classifier__splitter':
 ↪ splitters, 'classifier__max_depth': max_depths,␣
 ↪'classifier__min_samples_leaf' :                    min_samples_leafs}

    clf = GridSearchCV(pipe_svc, search_space, cv = StratifiedKFold(n_splits=5),
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_adult.append(clf.score(X_train, Y_train))
    metrics_test_adult.append(clf.score(X_test, Y_test))

print(metrics_train_adult)
print(metrics_test_adult)
```

```
[0.8532, 0.8692, 0.8628, 0.8514, 0.8698]
[0.8482275679402054, 0.8479735858640833, 0.8457240303327165, 0.8399912920431044,
0.8434019084938863]
```

[20]:
```
metrics_train_mush = []
metrics_test_mush = []

for x in range(5):
    X = mush_x
    Y = mush_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_svc = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪DecisionTreeClassifier())])
```

```
    criterions = ['gini']
    splitters = ['best']
    max_depths = [6]
    min_samples_leafs = [1]

    search_space = {'classifier__criterion': criterions, 'classifier__splitter':
 ↪ splitters, 'classifier__max_depth': max_depths,␣
 ↪'classifier__min_samples_leaf' :                  min_samples_leafs}

    clf = GridSearchCV(pipe_svc, search_space, cv = StratifiedKFold(n_splits=5),
                    verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_mush.append(clf.score(X_train, Y_train))
    metrics_test_mush.append(clf.score(X_test, Y_test))

print(metrics_train_mush)
print(metrics_test_mush)
```

```
[0.9996, 0.9994, 0.9988, 0.9996, 1.0]
[0.9996798975672215, 1.0, 0.9980793854033291, 0.9996798975672215, 1.0]
```

randomforests

March 17, 2021

[2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets
from sklearn import model_selection
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
```

[3]:
```
adult = pd.read_csv("adult.data", header = None)
mushroom = pd.read_csv("mushroom.data", header = None)
bank = pd.read_csv("bank.csv", delimiter = ";")
letterrecog1 = pd.read_csv("letterrecog.data", header = None)
letterrecog = pd.read_csv("letterrecog.data", header = None)
skin = pd.read_csv("skin.txt", delimiter = "\t", header = None)
```

[4]:
```
skin_y = skin[3]
skin_x = skin.drop(3, axis = 1)
skin.head(1000)

bank_y = bank["y"]
bank_y = bank_y.replace({'no': 0, 'yes': 1})
bank_x = bank.drop(["y"], axis = 1)
bank_x = pd.get_dummies(bank_x)
```

```
letterrecog[0] = letterrecog[0].replace(['A', 'B', 'C', 'D', 'E', 'F', 'G',␣
↪'H', 'I', 'J', 'K', 'L', 'M'], 0)
letterrecog[0] = letterrecog[0].replace(['N', 'O', 'P', 'Q', 'R', 'S', 'T',␣
↪'U', 'V', 'W', 'X', 'Y', 'Z'], 1)
letter_y = letterrecog[0]
letter_x = letterrecog.drop([0],axis=1)

letterrecog1[0] = letterrecog1[0].replace(['O'], 0)
letterrecog1[0] = letterrecog1[0].replace(['A', 'B', 'C', 'D', 'E', 'F', 'G',␣
↪'H', 'I', 'J', 'K', 'L', 'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',␣
↪'X', 'Y', 'Z'], 1)
letter1_y = letterrecog1[0]
letter1_x = letterrecog1.drop(0, axis = 1)

adult[14] = adult[14].replace({' <=50K': 0, ' >50K': 1})
adult_y = adult[14]
adult_x = adult.drop([14], axis = 1)
adult_x = pd.get_dummies(adult_x)

mushroom[0] = mushroom[0].replace({'p': 1, 'e': 0})
mush_y = mushroom[0]
mush_x = mushroom.drop(0, axis = 1)
mush_x = pd.get_dummies(mush_x)
```

[26]:
```
metrics_rf_skin = []
metrics_values_skin = []

for x in range(5):
    X = skin_x
    Y = skin_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
↪train_size = 5000, random_state = (10 * x))
    pipe_rf = Pipeline([('std', StandardScaler()), ('classifier',␣
↪RandomForestClassifier())])

    estimators = [1024]
    features = [1, 2]
    search_space = {'classifier__n_estimators': estimators,␣
↪'classifier__max_features': features}

    clf = GridSearchCV(pipe_rf, search_space, cv = StratifiedKFold(n_splits=5),
                       scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)
```

```
    metrics_rf_skin.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_rf_skin.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_rf_skin.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_skin.append(best_model.cv_results_['mean_test_accuracy'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_skin.append(best_model.cv_results_['mean_test_f1'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_skin.append(best_model.cv_results_['mean_test_roc_auc'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])
    print("done")
print(metrics_values_skin)
print(metrics_rf_skin)
```

```
done
done
done
done
done
[0.9978, 0.9948606984681767, 0.9999475037433412, 0.9978, 0.9946589843810472,
0.999931291060728, 0.9960000000000001, 0.9901037178268354, 0.9998788286260257,
0.9970000000000001, 0.9923376984360794, 0.9997515527950311, 0.9974000000000001,
0.9938444220301941, 0.9999470566244572]
[{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 2, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024}]
```

[27]:
```
metrics_rf_bank = []
metrics_values_bank = []

for x in range(5):
```

```
    X = bank_x
    Y = bank_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
↪train_size = 5000, random_state = (10 * x))
    pipe_rf = Pipeline([('std', StandardScaler()), ('classifier',␣
↪RandomForestClassifier())])

    estimators = [1024]
    features = [1, 2, 4, 6, 8, 12, 16, 20]
    search_space = {'classifier__n_estimators': estimators,␣
↪'classifier__max_features': features}

    clf = GridSearchCV(pipe_rf, search_space, cv = StratifiedKFold(n_splits=5),
                       scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_rf_bank.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_rf_bank.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_rf_bank.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_bank.append(best_model.cv_results_['mean_test_accuracy'][ np.
↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_bank.append(best_model.cv_results_['mean_test_f1'][ np.
↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_bank.append(best_model.cv_results_['mean_test_roc_auc'][ np.
↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])
    print("done")
print(metrics_values_bank)
print(metrics_rf_bank)
```

```
done
done
done
done
done
[0.9062000000000001, 0.5195929206742733, 0.9230956143697459, 0.8950000000000001,
0.5320656307329992, 0.9190762216072829, 0.8897999999999999, 0.4583136280475653,
0.9157669809309267, 0.9030000000000001, 0.4925667095025415, 0.9188698643299127,
0.9032, 0.5187430402256514, 0.9268298516110531]
[{'classifier__max_features': 20, 'classifier__n_estimators': 1024},
{'classifier__max_features': 20, 'classifier__n_estimators': 1024},
{'classifier__max_features': 12, 'classifier__n_estimators': 1024},
```

```
{'classifier__max_features': 8, 'classifier__n_estimators': 1024},
{'classifier__max_features': 20, 'classifier__n_estimators': 1024},
{'classifier__max_features': 12, 'classifier__n_estimators': 1024},
{'classifier__max_features': 4, 'classifier__n_estimators': 1024},
{'classifier__max_features': 20, 'classifier__n_estimators': 1024},
{'classifier__max_features': 16, 'classifier__n_estimators': 1024},
{'classifier__max_features': 8, 'classifier__n_estimators': 1024},
{'classifier__max_features': 20, 'classifier__n_estimators': 1024},
{'classifier__max_features': 16, 'classifier__n_estimators': 1024},
{'classifier__max_features': 20, 'classifier__n_estimators': 1024},
{'classifier__max_features': 20, 'classifier__n_estimators': 1024},
{'classifier__max_features': 12, 'classifier__n_estimators': 1024}]
```

[28]:
```
metrics_rf_letter = []
metrics_values_letter = []

for x in range(5):
    X = letter_x
    Y = letter_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
↪train_size = 5000, random_state = (10 * x))
    pipe_rf = Pipeline([('std', StandardScaler()), ('classifier',␣
↪RandomForestClassifier())])

    estimators = [1024]
    features = [1, 2, 4, 6, 8, 12, 16]
    search_space = {'classifier__n_estimators': estimators,␣
↪'classifier__max_features': features}

    clf = GridSearchCV(pipe_rf, search_space, cv = StratifiedKFold(n_splits=5),
                       scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_rf_letter.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_rf_letter.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_rf_letter.append(best_model.cv_results_['params'][ np.
↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_letter.append(best_model.cv_results_['mean_test_accuracy'][␣
↪np.argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_letter.append(best_model.cv_results_['mean_test_f1'][ np.
↪argmin(best_model.cv_results_['rank_test_f1']) ])
```

```
    metrics_values_letter.append(best_model.cv_results_['mean_test_roc_auc'][␣
 ↪np.argmin(best_model.cv_results_['rank_test_roc_auc']) ])
    print("done")
print(metrics_values_letter)
print(metrics_rf_letter)
```

```
done
done
done
done
done
[0.9366, 0.9370218477766936, 0.9880314827277659, 0.9364000000000001,
0.9363238205184388, 0.988378320018344, 0.9431999999999998, 0.9439610852934939,
0.9897006603886196, 0.9405999999999999, 0.9422262674813121, 0.9881748766643194,
0.943, 0.9431190644777562, 0.9891579055396047]
[{'classifier__max_features': 4, 'classifier__n_estimators': 1024},
{'classifier__max_features': 4, 'classifier__n_estimators': 1024},
{'classifier__max_features': 4, 'classifier__n_estimators': 1024},
{'classifier__max_features': 4, 'classifier__n_estimators': 1024},
{'classifier__max_features': 4, 'classifier__n_estimators': 1024},
{'classifier__max_features': 4, 'classifier__n_estimators': 1024},
{'classifier__max_features': 2, 'classifier__n_estimators': 1024},
{'classifier__max_features': 2, 'classifier__n_estimators': 1024},
{'classifier__max_features': 2, 'classifier__n_estimators': 1024},
{'classifier__max_features': 6, 'classifier__n_estimators': 1024},
{'classifier__max_features': 6, 'classifier__n_estimators': 1024},
{'classifier__max_features': 2, 'classifier__n_estimators': 1024},
{'classifier__max_features': 4, 'classifier__n_estimators': 1024},
{'classifier__max_features': 4, 'classifier__n_estimators': 1024},
{'classifier__max_features': 4, 'classifier__n_estimators': 1024}]
```

[29]:
```
metrics_rf_letter1 = []
metrics_values_letter1 = []

for x in range(5):
    X = letter1_x
    Y = letter1_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_rf = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪RandomForestClassifier())])

    estimators = [1024]
    features = [1, 2, 4, 6, 8, 12, 16]
    search_space = {'classifier__n_estimators': estimators,␣
 ↪'classifier__max_features': features}
```

```
    clf = GridSearchCV(pipe_rf, search_space, cv = StratifiedKFold(n_splits=5),
                       scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_rf_letter1.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_rf_letter1.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_rf_letter1.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_letter1.append(best_model.cv_results_['mean_test_accuracy'][␣
 ↪np.argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_letter1.append(best_model.cv_results_['mean_test_f1'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_letter1.append(best_model.cv_results_['mean_test_roc_auc'][␣
 ↪np.argmin(best_model.cv_results_['rank_test_roc_auc']) ])
    print("done")
print(metrics_values_letter1)
print(metrics_rf_letter1)
```

```
done
done
done
done
done
[0.9865999999999999, 0.9930631583446521, 0.996850072748968, 0.9865999999999999,
0.9930883569864738, 0.9953438576436598, 0.9855999999999998, 0.9925571297559198,
0.9958973769060673, 0.9870000000000001, 0.9932947573279286, 0.996170344346751,
0.9885999999999999, 0.9941207782016706, 0.9947505551443376]
[{'classifier__max_features': 4, 'classifier__n_estimators': 1024},
{'classifier__max_features': 4, 'classifier__n_estimators': 1024},
{'classifier__max_features': 2, 'classifier__n_estimators': 1024},
{'classifier__max_features': 4, 'classifier__n_estimators': 1024},
{'classifier__max_features': 4, 'classifier__n_estimators': 1024},
{'classifier__max_features': 2, 'classifier__n_estimators': 1024},
{'classifier__max_features': 8, 'classifier__n_estimators': 1024},
{'classifier__max_features': 8, 'classifier__n_estimators': 1024},
{'classifier__max_features': 2, 'classifier__n_estimators': 1024},
{'classifier__max_features': 4, 'classifier__n_estimators': 1024},
{'classifier__max_features': 4, 'classifier__n_estimators': 1024},
{'classifier__max_features': 2, 'classifier__n_estimators': 1024},
{'classifier__max_features': 12, 'classifier__n_estimators': 1024},
{'classifier__max_features': 12, 'classifier__n_estimators': 1024},
{'classifier__max_features': 4, 'classifier__n_estimators': 1024}]
```

[30]:
```
metrics_rf_adult = []
metrics_values_adult = []

for x in range(5):
    X = adult_x
    Y = adult_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_rf = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪RandomForestClassifier())])

    estimators = [1024]
    features = [1, 2, 4, 6, 8, 12, 16, 20]
    search_space = {'classifier__n_estimators': estimators,␣
 ↪'classifier__max_features': features}

    clf = GridSearchCV(pipe_rf, search_space, cv = StratifiedKFold(n_splits=5),
                       scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_rf_adult.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_rf_adult.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_rf_adult.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])

    metrics_values_adult.append(best_model.cv_results_['mean_test_accuracy'][␣
 ↪np.argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_adult.append(best_model.cv_results_['mean_test_f1'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_adult.append(best_model.cv_results_['mean_test_roc_auc'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])
    print("done")
print(metrics_values_adult)
print(metrics_rf_adult)
```

```
done
done
done
done
done
[0.8441999999999998, 0.650598117672549, 0.8968275575527966, 0.8576,
0.6628201278201278, 0.905629278241204, 0.8475999999999999, 0.6565093719845688,
0.9021123996302217, 0.8562, 0.6759646537354754, 0.900704612072961, 0.8624,
```

```
0.672995611064737, 0.9111500635290698]
[{'classifier__max_features': 12, 'classifier__n_estimators': 1024},
{'classifier__max_features': 12, 'classifier__n_estimators': 1024},
{'classifier__max_features': 20, 'classifier__n_estimators': 1024},
{'classifier__max_features': 20, 'classifier__n_estimators': 1024},
{'classifier__max_features': 20, 'classifier__n_estimators': 1024},
{'classifier__max_features': 20, 'classifier__n_estimators': 1024},
{'classifier__max_features': 12, 'classifier__n_estimators': 1024},
{'classifier__max_features': 12, 'classifier__n_estimators': 1024},
{'classifier__max_features': 20, 'classifier__n_estimators': 1024},
{'classifier__max_features': 16, 'classifier__n_estimators': 1024},
{'classifier__max_features': 16, 'classifier__n_estimators': 1024},
{'classifier__max_features': 20, 'classifier__n_estimators': 1024},
{'classifier__max_features': 20, 'classifier__n_estimators': 1024},
{'classifier__max_features': 20, 'classifier__n_estimators': 1024},
{'classifier__max_features': 20, 'classifier__n_estimators': 1024}]
```

[31]:

```
metrics_rf_mush = []
metrics_values_mush = []

for x in range(5):
    X = mush_x
    Y = mush_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_rf = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪RandomForestClassifier())])

    estimators = [1024]
    features = [1, 2, 4, 6, 8, 12, 16, 20]
    search_space = {'classifier__n_estimators': estimators,␣
 ↪'classifier__max_features': features}

    clf = GridSearchCV(pipe_rf, search_space, cv = StratifiedKFold(n_splits=5),
                       scoring = ['accuracy', 'roc_auc', 'f1'], refit = False,
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_rf_mush.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_rf_mush.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_rf_mush.append(best_model.cv_results_['params'][ np.
 ↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])
```

```
    metrics_values_mush.append(best_model.cv_results_['mean_test_accuracy'][ np.
↪argmin(best_model.cv_results_['rank_test_accuracy']) ])
    metrics_values_mush.append(best_model.cv_results_['mean_test_f1'][ np.
↪argmin(best_model.cv_results_['rank_test_f1']) ])
    metrics_values_mush.append(best_model.cv_results_['mean_test_roc_auc'][ np.
↪argmin(best_model.cv_results_['rank_test_roc_auc']) ])
    print("done")
print(metrics_values_mush)
print(metrics_rf_mush)
```

```
done
done
done
done
done
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
[{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024},
{'classifier__max_features': 1, 'classifier__n_estimators': 1024}]
```

[34]:
```
metrics_train_skin = []
metrics_test_skin = []

for x in range(5):
    X = skin_x
    Y = skin_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
↪RandomForestClassifier())])

    search_space = {'classifier__n_estimators': [1024]}

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
↪StratifiedKFold(n_splits=5),
```

```
                    verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_skin.append(clf.score(X_train, Y_train))
    metrics_test_skin.append(clf.score(X_test, Y_test))

print(metrics_train_skin)
print(metrics_test_skin)
```

```
[1.0, 1.0, 1.0, 1.0, 1.0]
[0.9974422741265616, 0.9976797177337049, 0.9975797414780656, 0.9973672919348321,
0.9978796702449835]
```

[35]:
```
metrics_train_mush = []
metrics_test_mush = []

for x in range(5):
    X = mush_x
    Y = mush_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪RandomForestClassifier())])

    search_space = {'classifier__n_estimators': [1024]}

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
 ↪StratifiedKFold(n_splits=5),
                    verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_mush.append(clf.score(X_train, Y_train))
    metrics_test_mush.append(clf.score(X_test, Y_test))

print(metrics_train_mush)
print(metrics_test_mush)
```

```
[1.0, 1.0, 1.0, 1.0, 1.0]
[1.0, 1.0, 1.0, 1.0, 1.0]
```

[36]:
```
metrics_train_letter = []
metrics_test_letter = []

for x in range(5):
    X = letter_x
```

```
    Y = letter_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪RandomForestClassifier())])

    search_space = {'classifier__n_estimators': [1024]}

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
 ↪StratifiedKFold(n_splits=5),
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_letter.append(clf.score(X_train, Y_train))
    metrics_test_letter.append(clf.score(X_test, Y_test))

print(metrics_train_letter)
print(metrics_test_letter)
```

```
[1.0, 1.0, 1.0, 1.0, 1.0]
[0.9439333333333333, 0.9491333333333334, 0.9465333333333333, 0.9454,
0.9489333333333333]
```

[37]:
```
metrics_train_letter1 = []
metrics_test_letter1 = []

for x in range(5):
    X = letter1_x
    Y = letter1_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪RandomForestClassifier())])

    search_space = {'classifier__n_estimators': [1024]}

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
 ↪StratifiedKFold(n_splits=5),
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_letter1.append(clf.score(X_train, Y_train))
    metrics_test_letter1.append(clf.score(X_test, Y_test))

print(metrics_train_letter1)
```

```
print(metrics_test_letter1)
```

```
[1.0, 1.0, 1.0, 1.0, 1.0]
[0.9895333333333334, 0.9884, 0.9902, 0.9881333333333333, 0.9878666666666667]
```

[38]:
```
metrics_train_bank = []
metrics_test_bank = []

for x in range(5):
    X = bank_x
    Y = bank_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪RandomForestClassifier())])

    search_space = {'classifier__n_estimators': [1024]}

    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
 ↪StratifiedKFold(n_splits=5),
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_bank.append(clf.score(X_train, Y_train))
    metrics_test_bank.append(clf.score(X_test, Y_test))

print(metrics_train_bank)
print(metrics_test_bank)
```

```
[1.0, 1.0, 1.0, 1.0, 1.0]
[0.90271318793365, 0.9025888438486981, 0.9007485513914103, 0.9009723707443237,
0.9004749944045162]
```

[5]:
```
metrics_train_adult = []
metrics_test_adult = []

for x in range(5):
    X = adult_x
    Y = adult_y
    X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪train_size = 5000, random_state = (10 * x))
    pipe_logreg = Pipeline([('std', StandardScaler()), ('classifier',␣
 ↪RandomForestClassifier())])

    search_space = {'classifier__n_estimators': [1024]}
```

```
    clf = GridSearchCV(pipe_logreg, search_space, cv =␣
↪StratifiedKFold(n_splits=5),
                       verbose = 0)

    best_model = clf.fit(X_train, Y_train)

    metrics_train_adult.append(clf.score(X_train, Y_train))
    metrics_test_adult.append(clf.score(X_test, Y_test))

print(metrics_train_adult)
print(metrics_test_adult)
```

```
[1.0, 1.0, 1.0, 1.0, 1.0]
[0.8526178295417438, 0.8508399550088894, 0.8517107506984507, 0.8485903994775226,
0.8496426109357426]
```

[ ]:

table 2 t test

March 18, 2021

[10]:
```
from scipy import stats
import numpy as np
import pandas as pd
```

[29]:
```
#accuracy
logreg_acc = [1, 1, 1, .999, .999, .954, .963, .961, .964, .965, .908, .926, .
 ↪916, .920, .922,
              .902, .892, .892, .902, .897, .735, .726, .723, .731, .732, .844,␣
 ↪.854, .848, .853, .859]

knn_acc = [.998, .997, .998, .997, .998, .890, .881, .883, .893, .889, .944, .
 ↪946, .950, .948, .945,
          .990, .990, .990, .990, .991, 1, 1, 1, .999, .999, .826, .833, .827, .
 ↪833, .842]

ann_acc = [0.9978, 0.9986, 0.9966000000000002, 0.9982, 0.9982000000000001, 0.
 ↪8988000000000002, 0.8914, 0.8922000000000001, 0.8996000000000001, 0.
 ↪8991999999999999, 0.9389999999999998, 0.9416, 0.9474, 0.9428000000000001, 0.
 ↪9414, 0.9944000000000001, 0.993, 0.9902000000000001, 0.992, 0.993, 0.8412, 0.
 ↪8505999999999998, 0.8413999999999999, 0.8516, 0.8562, 0.9998000000000001, 1.
 ↪0, 1.0, 0.9994,  0.9998000000000001]

dt_acc = [0.9906, 0.9872,0.985,0.9874,0.9864, 1.0, 1.0, 1.0, 0.9994, 0.
 ↪9998000000000001, 0.9018, 0.8855999999999999, 0.893, 0.8948, 0.
 ↪8996000000000001, 0.8074, 0.7882000000000001, 0.7796, 0.7914, 0.798, 0.9734,␣
 ↪0.9777999999999999, 0.9677999999999999, 0.9718,0.9736, 0.8406, 0.8652, 0.
 ↪8452, 0.8446, 0.8629999999999999]

rf_acc = [0.9978, 0.9978, 0.9960000000000001, 0.9970000000000001, 0.
 ↪9974000000000001, 0.9062000000000001, 0.8950000000000001, 0.
 ↪8897999999999999, 0.9030000000000001, 0.9032, 0.9366, 0.9364000000000001, 0.
 ↪9431999999999998, 0.9405999999999999, 0.943, 0.9865999999999999, 0.
 ↪9865999999999999, 0.9855999999999998, 0.9870000000000001, 0.
 ↪9885999999999999, 0.8441999999999998, 0.8576, 0.8475999999999999, 0.8562, 0.
 ↪8624, 1.0,  1.0, 1.0, 1.0, 1.0]
```

```
print(stats.ttest_rel(rf_acc, logreg_acc), stats.ttest_rel(rf_acc, knn_acc),␣
 ↪stats.ttest_rel(rf_acc, ann_acc), stats.ttest_rel(rf_acc, dt_acc))
```

```
Ttest_relResult(statistic=3.867728001330352, pvalue=0.0005722567958691182)
Ttest_relResult(statistic=0.23355496700863146, pvalue=0.8169713192819035)
Ttest_relResult(statistic=-2.4763255054895272e-14, pvalue=0.9999999999999805)
Ttest_relResult(statistic=1.36995338232576, pvalue=0.18121555984485413)
```

[30]:
```
#f1
logreg_f1 = [1, 1, 1, .999, .999, .979, .981, .979, .981, .982, .787, .825, .
 ↪795, .798, .818,
              .442, .472, .428, .467, .447, .733, .725, .723, .735, .730, .647, .
 ↪649, .647, .669, .663]

knn_f1 = [.996, .997, .994, .993, .994, .364, .392, .368, .400, .382, .945, .
 ↪946, .950, .949, .945,
          .995, .995, .995, .995, .995, 1, 1, 1, .999, .999, .597, .579, .578, .
 ↪613, .603]

ann_f1 = [0.9948824905581454, 0.9966054451277699, 0.9916170894593563, 0.
 ↪9954094482134043, 0.9957424347074116, 0.5038451747129866, 0.54058287345667,␣
 ↪0.5120690728958696, 0.5098754906106151, 0.5284074402802411, 0.
 ↪9393091457494618, 0.9414246260822754, 0.9478592912575454, 0.
 ↪9437818614638485, 0.9414850281626282, 0.9970831128699027, 0.996374718940962,␣
 ↪0.9949045327936945, 0.9958563062766084, 0.9963793249155086, 0.
 ↪6431564320208706, 0.6501685706776256, 0.6495924689453618, 0.
 ↪6654727666025264, 0.6604701264823458, 0.9997931747673215, 1.0, 1.0, 0.
 ↪9993820803295572, 0.9997901364113325]

dt_f1 = [0.9783551223596095, 0.9694808015693172, 0.9636832403125675, 0.
 ↪9685258257897829, 0.9683371724990139, 1.0, 0.9993820803295572, 1.0, 1.0, 0.
 ↪9997901364113325, 0.5127376773777481, 0.4446828338113097, 0.
 ↪4604738371483531, 0.44491646197578083,  0.47864127151463504, 0.
 ↪7951950913749362, 0.7801722213775234, 0.7775377192671089, 0.
 ↪7914780767487232, 0.7918010640923623, 0.9861674963117689, 0.
 ↪9885446069855736, 0.983248389737501, 0.9853404525761512, 0.9863523614671139,␣
 ↪0.6190663946269261, 0.6624313301171425, 0.6421086844109974, 0.
 ↪6410817899601222, 0.6562711112586372]
```

```
rf_f1 = [0.9948606984681767, 0.9946589843810472, 0.9901037178268354, 0.
 ↪9923376984360794, 0.9938444220301941, 0.5195929206742733, 0.
 ↪5320656307329992, 0.4583136280475653, 0.4925667095025415, 0.
 ↪5187430402256514, 0.9370218477766936, 0.9363238205184388, 0.
 ↪9439610852934939, 0.9422262674813121, 0.9431190644777562, 0.
 ↪9930631583446521, 0.9930883569864738, 0.9925571297559198, 0.
 ↪9932947573279286, 0.9941207782016706, 0.650598117672549, 0.6628201278201278,␣
 ↪0.6565093719845688, 0.6759646537354754, 0.672995611064737, 1.0, 1.0, 1.0, 1.
 ↪0, 1.0]

print(stats.ttest_rel(rf_f1, logreg_f1), stats.ttest_rel(rf_f1, knn_f1), stats.
 ↪ttest_rel(rf_f1, ann_f1), stats.ttest_rel(rf_f1, dt_f1))
```

```
Ttest_relResult(statistic=1.3196668649702783, pvalue=0.19727003894795328)
Ttest_relResult(statistic=0.7508532154953843, pvalue=0.45879076329301727)
Ttest_relResult(statistic=-0.8641945476459633, pvalue=0.39457306415758286)
Ttest_relResult(statistic=0.6152584301890378, pvalue=0.5431825105351522)
```

[31]:
```
#rocauc
logreg_roc = [1, 1, 1, .999, .999, .858, .846, .845, .864, .842, .947, .957, .
 ↪949, .952, .953,
              .908, .897, .898, .897, .908, .818, .811, .815, .817, .817, .899,␣
 ↪.904, .900, .903, .910]

knn_roc = [.999, .999, .999, .999, .999, .857, .855, .846, .842, .851, .983, .
 ↪983, .985, .986, .981,
           .995, .993, .993, .995, .993, 1, 1, 1, .999, .999, .873, .878, .868, .
 ↪874, .877]

ann_roc = [0.999923641808496,0.9999006910529529,0.9997489398876847,0.
 ↪9995973881191272,0.9999686809734742, 0.9021220811290803,0.8968738464234173,␣
 ↪0.9027597778828416, 0.8916224681963181, 0.9056856336793313, 0.
 ↪9853412535377963, 0.9873919198744044, 0.9880109204756881, 0.
 ↪9874993186539631, 0.9861750824241332, 0.9982729384436702, 0.
 ↪9972474025194387, 0.9976562316563943, 0.9965002895050012, 0.
 ↪9965891931902295, 0.8865650916525893, 0.8890645991749528, 0.
 ↪8860342180864638, 0.8928723260287585, 0.8928082374838995, 1.0, 1.0, 1.0, 1.
 ↪0, 1.0]
```

```
dt_roc = [0.9929714670914092, 0.991552542390532, 0.9894727555958157, 0.
 ↪9926803631151457, 0.9916108204492871, 1.0, 1.0, 1.0, 1.0, 0.
 ↪9999995991518051, 0.8514674480383858, 0.847997518145152, 0.8561809939650138,␣
 ↪0.8477093662323686, 0.8540245104434039, 0.8927800956794988, 0.
 ↪8809747849609169, 0.8730566744318441, 0.8810694100842348, 0.
 ↪8928823012944813, 0.9459189106650092, 0.9350418735620096, 0.
 ↪9309169668347608, 0.9267867697553683, 0.9432301998519614, 0.
 ↪8820078832707374, 0.8945836262491371, 0.8879380735839792, 0.
 ↪8796378332289819, 0.8970064564357567]

rf_roc = [0.9999475037433412, 0.999931291060728, 0.9998788286260257, 0.
 ↪9997515527950311, 0.9999470566244572, 0.9230956143697459, 0.
 ↪9190762216072829, 0.9157669809309267, 0.9188698643299127, 0.
 ↪9268298516110531, 0.9880314827277659, 0.988378320018344, 0.9897006603886196,␣
 ↪0.9881748766643194, 0.9891579055396047, 0.996850072748968, 0.
 ↪9953438576436598, 0.9958973769060673, 0.996170344346751, 0.9947505551443376,␣
 ↪0.8968275575527966, 0.905629278241204, 0.9021123996302217, 0.
 ↪900704612072961, 0.9111500635290698, 1.0, 1.0, 1.0, 1.0, 1.0]

print(stats.ttest_rel(rf_roc, logreg_roc), stats.ttest_rel(rf_roc, knn_roc),␣
 ↪stats.ttest_rel(rf_roc, ann_roc), stats.ttest_rel(rf_roc, dt_roc))
```

```
Ttest_relResult(statistic=9.75905802937096, pvalue=1.1443704706062473e-10)
Ttest_relResult(statistic=1.4145924424865954, pvalue=0.1678359253333242)
Ttest_relResult(statistic=3.579083364747849, pvalue=0.001237508225403238)
Ttest_relResult(statistic=2.8032946027794803, pvalue=0.00892666804251882)
```

[32]:
```
#mean
logreg_mean = logreg_acc + logreg_f1 + logreg_roc
knn_mean = knn_acc + knn_f1 + knn_roc
ann_mean = ann_acc + ann_f1 + ann_roc
dt_mean = dt_acc + dt_f1 + dt_roc
rf_mean = rf_acc + rf_f1 + rf_roc

print(stats.ttest_rel(rf_mean, logreg_mean), stats.ttest_rel(rf_mean,␣
 ↪knn_mean), stats.ttest_rel(rf_mean, ann_mean), stats.ttest_rel(rf_mean,␣
 ↪dt_mean))
```

```
Ttest_relResult(statistic=3.2099361252742082, pvalue=0.001847190611251427)
Ttest_relResult(statistic=1.1550049028220701, pvalue=0.25118005708869684)
Ttest_relResult(statistic=1.3302322794367825, pvalue=0.18684044855153442)
Ttest_relResult(statistic=1.6189398343373622, pvalue=0.10899870294043103)
```

[ ]:
```

```

table 3 t test

March 18, 2021

[8]:
```
from scipy import stats
import numpy as np
import pandas as pd
```

[9]:
```
#skin
logreg_skin = [0.9075999999999999, 0.786988622695977, 0.9472877928306817, 0.
 ↪9263999999999999, 0.8251045790195587, 0.9565529026394944, 0.
 ↪9162000000000001, 0.7946894246004156, 0.9487275729152576, 0.
 ↪9198000000000001, 0.7983778935234352, 0.9520484153527631, 0.9218, 0.
 ↪817530089865848, 0.9527263798494608]
knn_skin = [0.9984, 0.9962746621603266, 0.9993962930484217, 0.9986, 0.
 ↪9966054508474391, 0.9996669150756784, 0.9976, 0.9940715631932495, 0.
 ↪9992869587566673, 0.9972, 0.9928701185087376, 0.9992221691352127, 0.9976, 0.
 ↪9943060418097286, 0.9997857273583002]
ann_skin = [0.9978, 0.9948824905581454, 0.999923641808496, 0.9986, 0.
 ↪9966054451277699, 0.9999006910529529, 0.9966000000000002, 0.
 ↪9916170894593563, 0.9997489398876847, 0.9982, 0.9954094482134043, 0.
 ↪9995973881191272, 0.9982000000000001, 0.9957424347074116, 0.9999686809734742]
dt_skin = [0.9906, 0.9783551223596095, 0.9929714670914092, 0.9872, 0.
 ↪9694808015693172, 0.991552542390532, 0.985, 0.9636832403125675, 0.
 ↪9894727555958157, 0.9874, 0.9685258257897829, 0.9926803631151457, 0.9864, 0.
 ↪9683371724990139, 0.9916108204492871]
rf_skin = [0.9978, 0.9948606984681767, 0.9999475037433412, 0.9978, 0.
 ↪9946589843810472, 0.999931291060728, 0.9960000000000001, 0.9901037178268354,␣
 ↪0.9998788286260257, 0.9970000000000001, 0.9923376984360794, 0.
 ↪9997515527950311, 0.9974000000000001, 0.9938444220301941, 0.9999470566244572]

print(stats.ttest_rel(rf_skin, logreg_skin), stats.ttest_rel(rf_skin,␣
 ↪knn_skin), stats.ttest_rel(rf_skin, ann_skin), stats.ttest_rel(rf_skin,␣
 ↪dt_skin))
```

```
Ttest_relResult(statistic=6.471083156756916, pvalue=1.4704252010095685e-05)
Ttest_relResult(statistic=-2.0523536827342577, pvalue=0.05932231963725921)
Ttest_relResult(statistic=-3.0474907715100605, pvalue=0.00869297472782845)
Ttest_relResult(statistic=7.180900563209188, pvalue=4.702698378909285e-06)
```

[10]:
```
#mushroom
logreg_mush = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.9994, 0.
 ↪9993820803295572, 0.9999295523774071, 0.9998000000000001, 0.
 ↪9997901364113325, 0.9999759491083132]
knn_mush = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.9994, 0.
 ↪9993820803295572, 0.9998326766429775, 0.9998000000000001, 0.
 ↪9997901364113325, 0.9999931849764208]
ann_mush = [0.9998000000000001, 0.9997931747673215, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
 ↪0, 1.0, 0.9994, 0.9993820803295572, 1.0, 0.9998000000000001, 0.
 ↪9997901364113325, 1.0]
dt_mush = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.9994, 0.
 ↪9993820803295572, 1.0, 0.9998000000000001, 0.9997901364113325, 0.
 ↪9999995991518051]
rf_mush = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
 ↪0, 1.0]

print(stats.ttest_rel(rf_mush, logreg_mush), stats.ttest_rel(rf_mush,␣
 ↪knn_mush), stats.ttest_rel(rf_mush, ann_mush), stats.ttest_rel(rf_mush,␣
 ↪dt_mush))
```

```
Ttest_relResult(statistic=2.0902343283902503, pvalue=0.05532302202147373)
Ttest_relResult(statistic=2.1803759238962264, pvalue=0.04678804784264118)
Ttest_relResult(statistic=2.4712180951397182, pvalue=0.0269210898466897)
Ttest_relResult(statistic=1.9523346480147878, pvalue=0.07119128303858528)
```

[11]:
```
#adult
logreg_adult = [0.8443999999999999, 0.6465204019598264, 0.8993221043193984, 0.
 ↪8542, 0.6493796243856014, 0.9038224910142307, 0.8482, 0.6471540625387119, 0.
 ↪9003911031874621, 0.8526, 0.6692540979018624, 0.9027059802301747, 0.
 ↪8591999999999999, 0.6629474438093611, 0.909544651608934]
knn_adult = [0.826, 0.5974798034133271, 0.8733100443365643, 0.833, 0.
 ↪578699020319533, 0.8784279370870373, 0.827, 0.5778683664585215, 0.
 ↪8677767069695934, 0.8336, 0.612577561553796, 0.8743990826673282, 0.
 ↪8421999999999998, 0.603276595593697, 0.8769401736417425]
ann_adult = [0.8412, 0.6431564320208706, 0.8865650916525893, 0.
 ↪8505999999999998, 0.6501685706776256, 0.8890645991749528, 0.
 ↪8413999999999999, 0.6495924689453618, 0.8860342180864638, 0.8516, 0.
 ↪6654727666025264, 0.8928723260287585, 0.8562, 0.6604701264823458, 0.
 ↪8928082374838995]
dt_adult = [0.8406, 0.6190663946269261, 0.8820078832707374, 0.8652, 0.
 ↪6624313301171425, 0.8945836262491371, 0.8452, 0.6421086844109974, 0.
 ↪8879380735839792, 0.8446, 0.6410817899601222, 0.8796378332289819, 0.
 ↪8629999999999999, 0.6562711112586372, 0.8970064564357567]
```

```
rf_adult = [0.8441999999999998, 0.650598117672549, 0.8968275575527966, 0.8576,␣
 ↪0.6628201278201278, 0.905629278241204, 0.8475999999999999, 0.
 ↪6565093719845688, 0.9021123996302217, 0.8562, 0.6759646537354754, 0.
 ↪900704612072961, 0.8624, 0.672995611064737, 0.9111500635290698]

print(stats.ttest_rel(rf_adult, logreg_adult), stats.ttest_rel(rf_adult,␣
 ↪knn_adult), stats.ttest_rel(rf_adult, ann_adult), stats.ttest_rel(rf_adult,␣
 ↪dt_adult))
```

```
Ttest_relResult(statistic=3.0293317186409423, pvalue=0.009011853424271382)
Ttest_relResult(statistic=6.695377921119806, pvalue=1.0181337485397777e-05)
Ttest_relResult(statistic=8.16137975634417, pvalue=1.0845446371937279e-06)
Ttest_relResult(statistic=4.0748413539814905, pvalue=0.0011366585217843395)
```

[12]:
```
#letter1
logreg_letter1 = [0.7346, 0.733481022102547, 0.817850551040127, 0.726, 0.
 ↪7249284357533791, 0.811482000048866, 0.723, 0.7229849503673251, 0.
 ↪8152916550007534, 0.7305999999999999, 0.7350522116576401, 0.
 ↪8166769774235945, 0.7322, 0.7298177422174373, 0.8172815717999301]
knn_letter1 = [0.9443999999999999, 0.9448772908157576, 0.9834827179290511, 0.
 ↪9458, 0.9457846476329509, 0.9834278938565377, 0.9501999999999999, 0.
 ↪9504897097403239, 0.9846602027714292, 0.9479999999999998, 0.
 ↪9492630553423774, 0.9855071338903029, 0.9448000000000001, 0.
 ↪9448245378332271, 0.9812550319610389]
ann_letter1 = [0.9389999999999998, 0.9393091457494618, 0.9853412535377963, 0.
 ↪9416, 0.9414246260822754, 0.9873919198744044, 0.9474, 0.9478592912575454, 0.
 ↪9880109204756881, 0.9428000000000001, 0.9437818614638485, 0.
 ↪9874993186539631, 0.9414, 0.9414850281626282, 0.9861750824241332]
dt_letter1 = [0.8074, 0.7951950913749362, 0.8927800956794988, 0.
 ↪7882000000000001, 0.7801722213775234, 0.8809747849609169, 0.7796, 0.
 ↪7775377192671089, 0.8730566744318441, 0.7914, 0.7914780767487232, 0.
 ↪8810694100842348, 0.798, 0.7918010640923623, 0.8928823012944813]
rf_letter1 = [0.9366, 0.9370218477766936, 0.9880314827277659, 0.
 ↪9364000000000001, 0.9363238205184388, 0.988378320018344, 0.9431999999999998,␣
 ↪0.9439610852934939, 0.9897006603886196, 0.9405999999999999, 0.
 ↪9422262674813121, 0.9881748766643194, 0.943, 0.9431190644777562, 0.
 ↪9891579055396047]

print(stats.ttest_rel(rf_letter1, logreg_letter1), stats.ttest_rel(rf_letter1,␣
 ↪knn_letter1), stats.ttest_rel(rf_letter1, ann_letter1), stats.
 ↪ttest_rel(rf_letter1, dt_letter1))
```

```
Ttest_relResult(statistic=39.90618376002439, pvalue=8.026277183501743e-16)
Ttest_relResult(statistic=-1.7133608081947316, pvalue=0.10869220786950445)
Ttest_relResult(statistic=-1.3102598897219455, pvalue=0.21119859709329755)
Ttest_relResult(statistic=21.493478027574252, pvalue=4.042339405345073e-12)
```

[13]:
```
#letter2
logreg_letter2 = [0.9593999999999999, 0.9792793074350721, 0.8580654361864918, 0.
 ↪9632, 0.9812550514456342, 0.846165243655258, 0.9606, 0.9799040472894921, 0.
 ↪8451457928940206, 0.9635999999999999, 0.9814625562727153, 0.
 ↪8642619599278965, 0.9650000000000001, 0.9821882951653944, 0.8424041450777203]
knn_letter2 = [0.9904, 0.9949859552330796, 0.9945953570474835, 0.9902, 0.
 ↪9949064810363015, 0.9928141250083993, 0.9904, 0.9949978062967325, 0.
 ↪993187745345727, 0.9896, 0.9945963156421588, 0.9945767560032985, 0.9906, 0.
 ↪9951233371257567, 0.9932760917838639]
ann_letter2 = [0.9944000000000001, 0.9970831128699027, 0.9982729384436702, 0.
 ↪993, 0.996374718940962, 0.9972474025194387, 0.9902000000000001, 0.
 ↪9949045327936945, 0.9976562316563943, 0.992, 0.9958563062766084, 0.
 ↪9965002895050012, 0.993, 0.9963793249155086, 0.9965891931902295]
dt_letter2 = [0.9734, 0.9861674963117689, 0.9459189106650092, 0.
 ↪9777999999999999, 0.9885446069855736, 0.9350418735620096, 0.
 ↪9677999999999999, 0.983248389737501, 0.9309169668347608, 0.9718, 0.
 ↪9853404525761512, 0.9267867697553683, 0.9736, 0.9863523614671139, 0.
 ↪9432301998519614]
rf_letter2 = [0.9865999999999999, 0.9930631583446521, 0.996850072748968, 0.
 ↪9865999999999999, 0.9930883569864738, 0.9953438576436598, 0.
 ↪9855999999999998, 0.9925571297559198, 0.9958973769060673, 0.
 ↪9870000000000001, 0.9932947573279286, 0.996170344346751, 0.9885999999999999,␣
 ↪0.9941207782016706, 0.9947505551443376]

print(stats.ttest_rel(rf_letter2, logreg_letter2), stats.ttest_rel(rf_letter2,␣
 ↪knn_letter2), stats.ttest_rel(rf_letter2, ann_letter2), stats.
 ↪ttest_rel(rf_letter2, dt_letter2))
```

```
Ttest_relResult(statistic=3.7829612397119976, pvalue=0.0020175747959176103)
Ttest_relResult(statistic=-1.5343354618324656, pvalue=0.14723210235695683)
Ttest_relResult(statistic=-6.361040196953286, pvalue=1.7654664781585676e-05)
Ttest_relResult(statistic=4.263004735983468, pvalue=0.0007880224751073547)
```

[14]:
```
#bank
logreg_bank = [0.9022, 0.44217572133729444, 0.9076338923490169, 0.8924, 0.
 ↪4724843378073258, 0.8974646732067194, 0.8916000000000001, 0.427716898262386,␣
 ↪0.8979985352875193, 0.9022, 0.46673204419425024, 0.8971463161535802, 0.8974,␣
 ↪0.4465196692274261, 0.9077215380221701]
knn_bank = [0.8904, 0.36384401503290537, 0.8569548458093218, 0.8808, 0.
 ↪3917163936203515, 0.8553697767492773, 0.8831999999999999, 0.
 ↪36831698586081646, 0.8462858414216982, 0.8934, 0.40028555845748925, 0.
 ↪8421278680963911, 0.8886, 0.381534064409389, 0.8513226181493809]
ann_bank = [0.8988000000000002, 0.5038451747129866, 0.9021220811290803, 0.8914,␣
 ↪0.54058287345667, 0.8968738464234173, 0.8922000000000001, 0.
 ↪5120690728958696, 0.9027597778828416, 0.8996000000000001, 0.
 ↪5098754906106151, 0.8916224681963181, 0.8991999999999999, 0.
 ↪5284074402802411, 0.9056856336793313]
```

```
dt_bank = [0.9018, 0.5127376773777481, 0.8514674480383858, 0.8855999999999999,␣
 ↪0.4446828338113097, 0.847997518145152, 0.893, 0.4604738371483531, 0.
 ↪8561809939650138, 0.8948, 0.44491646197578083, 0.8477093662323686, 0.
 ↪8996000000000001, 0.47864127151463504, 0.8540245104434039]
rf_bank = [0.9062000000000001, 0.5195929206742733, 0.9230956143697459, 0.
 ↪8950000000000001, 0.5320656307329992, 0.9190762216072829, 0.
 ↪8897999999999999, 0.4583136280475653, 0.9157669809309267, 0.
 ↪9030000000000001, 0.4925667095025415, 0.9188698643299127, 0.9032, 0.
 ↪5187430402256514, 0.9268298516110531]

print(stats.ttest_rel(rf_bank, logreg_bank), stats.ttest_rel(rf_bank,␣
 ↪knn_bank), stats.ttest_rel(rf_bank, ann_bank), stats.ttest_rel(rf_bank,␣
 ↪dt_bank))
```

```
Ttest_relResult(statistic=3.791320729999022, pvalue=0.0019845334595098214)
Ttest_relResult(statistic=5.338848836343396, pvalue=0.00010454232005222095)
Ttest_relResult(statistic=0.5947304413943297, pvalue=0.561513245058838)
Ttest_relResult(statistic=4.214781344644656, pvalue=0.0008653244268273456)
```

[15]:
```
#mean
logreg_mean = logreg_skin + logreg_mush + logreg_adult + logreg_letter1 +␣
 ↪logreg_letter2 + logreg_bank
knn_mean = knn_skin + knn_mush + knn_adult + knn_letter1 + knn_letter2 +␣
 ↪knn_bank
ann_mean = ann_skin + ann_mush + ann_adult + ann_letter1 + ann_letter2 +␣
 ↪ann_bank
dt_mean = dt_skin + dt_mush + dt_adult + dt_letter1 + dt_letter2 + dt_bank
rf_mean = rf_skin + rf_mush + rf_adult + rf_letter1 + rf_letter2 + rf_bank

print(stats.ttest_rel(rf_mean, logreg_mean), stats.ttest_rel(rf_mean,␣
 ↪knn_mean), stats.ttest_rel(rf_mean, ann_mean), stats.ttest_rel(rf_mean,␣
 ↪dt_mean))
```

```
Ttest_relResult(statistic=7.833992382809698, pvalue=9.504983362537794e-12)
Ttest_relResult(statistic=4.68962711431836, pvalue=9.822936754066564e-06)
Ttest_relResult(statistic=1.3302322794367827, pvalue=0.18684044855153434)
Ttest_relResult(statistic=7.170596725713598, pvalue=2.1080841709363569e-10)
```

[ ]: