Pooja Pathak

Professor Jason Fleischer

COGS 118A

March 18, 2021

COGS 118A: Comparison of Binary Classification Models

Abstract

This paper attempts to replicate the work done by Caruana and Niculescu-Mizil in their 2006 paper titled "An Empirical Comparison of Supervised Machine Learning Algorithms" and analyzes the binary classification performance of three supervised learning algorithms over four different datasets (Caruana and Niculescu-Mizil). The algorithms described and used in this paper are Logistic Regression, Random Forests, and k-Nearest Neighbors. The optimal hyperparameters for each algorithm are found by tuning different sets of hyperparameters by grid searches, based on the three scoring metrics - accuracy, ROC Area under Curve and F-score. The results of this paper are consistent with Caruana's 2006 analysis (referred to as CNM06) that random forests have the overall best average performance over three metrics and four datasets of different balances as compared to Logistic Regression and k-Nearest Neighbors. We also observe that random forests and k-Nearest neighbors performed similarly across the four problem sets when compared across the three metrics.
Keywords: *CNM06* - Caruana Paper, *LR* - Logistic Regression, *RF* - Random Forests, *kNN* - k-Nearest Neighbors

1. Introduction

The CNM06 paper is an empirical comparison study done on the binary classification performance of ten supervised learning algorithms across eleven different datasets using eight scoring metrics. Conducted as a follow-up to STATLOG after new learning algorithms emerged, the main aim of CNM06 was to evaluate, compare and contrast whether certain algorithms performed better on certain datasets using certain metrics. The paper provides a comprehensive review of each algorithm's performance score on each metric, for each problem and their averages. Caruana and Niculescu-Mizil (2006) ultimately concluded that on average random forests and boosted trees perform well, while logistic regression, stumps, naive bayes and decision trees perform poorly. However, they caution that while it is possible to make some generalizations regarding algorithm performance, certain algorithms perform better on some test sets than others.

This project attempts to reproduce the analysis done by the CNM06 paper using only three algorithms (Logistic Regression, Random Forests, k-Nearest Neighbors), four of the original problem sets used in CNM06 paper (COV_TYPE, ADULT, LETTER.p1, LETTER.p2) across three metrics. Since accuracy alone does not provide a complete idea as to whether a classifier performs well (since a poor classifier can predict with high accuracy on unbalanced data), we evaluate the performance of each classifier on ROC area and F-score as well for a more thorough analysis of the classifier performance across problem sets.

We would like to examine whether the results of CNM06 hold true across a smaller subset of algorithms, problems and metrics, as well as explore and evaluate the differences if any. For each algorithm, we explore variations in hyperparameters and conduct a Grid Search to find the best hyperparameters per metric for each model. It should be noted that we do not calibrate the predicted results of each classifier, as done in the original study.
The results of these Experiments are consistent with CNM06, and are outlined in the Experiment and Results section.

2. Methodology

2.1. Learning Algorithms

The hyperparameters selected for the three algorithms follow those presented in CNM06 and are laid out as follows -

Logistic Regression (LR): Both regularized (Ridge (L1) and Lasso (L2) regularization) and unregularized models are used. For each regularized model, the regularization strength parameter C is varied by factors of 10 from 10^{-8} to 10^{4}. For the unregularized model, the penalty parameter is set to none. Two different solvers namely saga and sag are used, with the max_iter parameter set to 10000.

Random Forests (RF): The forests have 1024 total tree estimators. The possible max number of features per tree split is as follows: 1, 2, 4, 6, 8, 12, 16.

k-Nearest Neighbors (kNN): The different hyperparameters for this classifier are the number of neighbors and types of weights.
Number of neighbors : 26 equally spaced (4-step) values between 1 and 500, starting from 1.
[1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49, 53, 57, 61, 65, 69, 73, 77, 81, 85, 89, 93, 97, 101]
Types of weights varied : [uniform, distance].

2.2 Performance Metrics

The three scoring metrics used to evaluate the performance of each classifier during Grid Search, 5-fold cross validation and on the train and test set are accuracy (ACC), area under ROC curve (ROC), and F-score (FSC). Each metric lies within the range [0, 1].

Accuracy:
The threshold metric of accuracy (ACC) is calculated using the following equation:

$$Accuracy = (\Sigma\, TruePositive + \Sigma\, TrueNegative)\, \frac{1}{TruePositive + FalsePositive + TrueNegative + FalseNegative}$$

F-Score:

The F-score metric depends on the values of two other metrics -Precision and Recall.
Precision is defined as the number of true positive cases divided by the sum of true positives and false positives. It is a rank metric that can help us understand how accurately our classifier can predict positive cases from the positive space.
Recall/Sensitivity is defined as the number of true positive cases divided by the sum of true positives and false negatives. It represents the proportion of positive cases that were accurately classified out of the

total. Thus, F-score incorporates these two metrics and gives us a good idea of the balance between the two (Koehrsen 3).
It is calculated using the following equation:

$$F-score = (2 * Precision * Recall)\frac{1}{(Precision + Recall)} = \frac{True\ Positives}{True\ Positive + 0.5\ (False\ Positive + False\ Negative)}$$

ROC Area Under Curve:

The ROC curve is a probability curve plotted between the True Positive Rate (TPR) on the X-axis, and the False Positive Rate (FPR) on the Y-axis. Here, TPR refers to Recall/Sensitivity as defined in the F-score section above. FPR is calculated as 1 - Specificity, where Specificity is the number of false positive cases divided by the sum of true negatives and false positives (Narkhede 2).
The area under this ROC curve (AUC) measures how well the classifier can separate the positive cases from the negative cases.

2.3 Datasets

The four datasets used in this analysis are available online through the UCI Machine Learning Repository. All integer continuous data is standardized to mean 0 and unit variance.

ADULT: This dataset contains categorical data, so one-hot encoding is used to convert these categories to integers. After one-hot encoding, there are 105 columns in total to represent the 14 main features of the problem set.
Binary Classification Task: The target variable "target_income" is converted to binary labels as follows-
(> 50K) is represented as 1. (Positive class)
(<=50K) is represented as 0. (Negative class)
This dataset contains 24.08% positive labels and 75.91% negative labels so it is slightly unbalanced.

COV_TYPE: This dataset is already converted to integer values, so no more data cleaning or processing is done. There are 13 main features in this set.
Binary Classification Task: The target variable "forest_cover" is converted to binary labels as follows-
The largest class in the dataset is represented as 1. (Positive class)
All other classes are represented as 0. (Negative class).
This dataset contains 51.24% positive labels and 48.76% negative labels, so it is relatively well-balanced.

LETTER: We perform two different binary classification tasks on the LETTER dataset. For the purpose of analysis, it is split into problem sets **LETTER.p1** and **LETTER.p2**, with different distribution of positive and negative labels. There are 17 main features in this set.
For **LETTER.p1**,
Binary Classification Task: The target variable "letter" is converted to binary labels as follows-
The letter 'O' is represented as 1. (Positive class)
All other letters are represented as 0. (Negative class).
This dataset contains 3.76% positive labels and 96.24% negative labels, so it is extremely unbalanced.
For **LETTER.p2,**

Binary Classification Task: The target variable “letter” is converted to binary labels as follows-
All letters between A-M are represented as 1. (Positive class)
The remaining letters are represented as 0. (Negative class).
This dataset contains 49.7% positive labels and 50.3% negative labels, so it is well-balanced.

Table 1: Problem Set Description

	#ATTR	TRAIN SIZE	TEST SIZE	%POZ
COV_TYPE	54	5000	576012	51%
ADULT	14/105	5000	27561	24%
LETTER.p1	16	5000	15000	4%
LETTER.p2	16	5000	15000	50%

Table 1 lists the overall set-up of each dataset: the number of attributes, train size, test size and percentage of positive labels per set. From the above table we observe that COV_TYPE and LETTER.p2 are well-balanced datasets, whereas ADULT and LETTER.p1 are unbalanced.

Experiment and Results

Experiment set-up:

As mentioned above, this experiment involves running three algorithms over four different datasets. For each algorithm and dataset, we first split our data into training and test sets. For our training set, we randomly select 5000 samples with replacement. For our test set, we use all other remaining data samples other than the 5000 chosen for the training set. Once the train-test split is done, we conduct a stratified five-fold Grid Search with the chosen classifier, over different variations of hyperparameters to obtain the best classifier. The optimal hyperparameters for ACC, FSC and ROC are then extracted from the cv_results of this best classifier. Next, we train three models each tuned for a different metric over the entire training set. These three models are then used to make predictions on the test set, each model yielding a performance score for that metric. Thus, for a single classifier over a single dataset, we obtain three test scores for each metric- ACC, ROC and FSC. This process is repeated for five trials and for each dataset-algorithm combination. So, overall for a single classifier under a single performance metric, we obtain 4 datasets x 5 trials = 20 total test scores.

The following paragraphs discuss the results and observations from Tables 2, 3, 4 and 5.

Table 2: Performance over metrics (averaged over four problems)

MODEL	ACC	FSC	ROC	MEAN
LR	0.763	0.598	0.779	0.713
RF	**0.902**	**0.816**	0.852*	**0.859**

KNN	0.889*	0.805*	**0.853**	0.853

In Table 2, the three rows list the different classifiers used, and the columns are the three different metrics used for comparison. Each entry in the table is the average ACC, FSC and ROC score per classifier across all four datasets for five trials. (See Table 11 in Appendix for raw test scores per trial). The ACC, FSC and ROC score of the best-performing classifier in each column are bold-faced. The Asterisk symbol (*) represent scores that are not statistically significant/different from the best performance score in that metric column. In other words, all classifiers with performance scores marked with (*) performed about the same as the best performing classifier for that metric. These annotation comparisons are done in Appendix Table 6, where independent two-sample t-tests are conducted between the best classifier per problem and each of the remaining classifiers. The significance threshold for the t-tests is set to 0.05.

We observe that random forests had the highest performance scores across all four datasets for the metrics ACC and FSC across five trials. Further t-test comparisons reveal that there is no significant difference between the performance of random forests and k-Nearest Neighbors (kNN) for ACC and FSC metrics ($p = 0.616$ for ACC, $p = 0.770$ for FSC). Interestingly enough, k-NNs had the highest performance score for the ROC metric across 4 problems and 5 trials, which was also not statistically different from RF performance for the ROC metric. (See Table 6 in Appendix). These results imply that RFs and k-NNs had similar performances for all metrics across 4 differently balanced datasets. We also have evidence that Logistic Regression (LR) performed poorly in each column metric which aligns with the CNM06 Table 2 results.

In Table 3, the three rows list the different classifiers used, while the columns list the datasets used for comparison. Each entry in the table is the average performance score across the three different metrics across 5 different trials.

Table 3: Performance by problem (averaged over three metrics)

MODEL	COV_TYPE	ADULT	LETTER.p1	LETTER.p2	MEAN
LR	0.752	**0.770**	0.603	0.728	0.713
RF	**0.821**	0.761*	0.898*	0.946	0.884
KNN	0.778	0.734*	**0.933**	**0.953**	**0.888**

As CNM06 suggests, per the No Free Lunch Theorem, there is no one learning algorithm that performs optimally well across all datasets. This is reflected in the Table 3 results where we observe that RF performs well across COV_TYPE, ADULT, and LETTER.p1 but not across LETTER.p2.
Under the balanced COV_TYPE dataset, we observe that RF performed the best across all metrics and all 5 trials, which corroborates the results for COV_TYPE column in CNM06 Table 3.
Under the relatively unbalanced ADULT dataset, we observe that LR performs the best, but further t-test comparisons reveal that this difference in average performance score between LR, RF and KNN is not

significant (See Appendix Table 7). Thus, it is likely that all three classifiers have similar performance on the ADULT dataset, and LR had the highest average performance score by chance.
Under the extremely unbalanced LETTER.p1 dataset, kNNs perform best with insignificant differences when compared to RF ($p = 0.139$). LRs perform significantly worse, which follows the same results as LETTER.p1 column in Table 3 of CNM06.
Interestingly enough, under the relatively balanced LETTER.p2 dataset, kNN once again had the best performance (0.953), while RF had a similar average performance score (0.946).
However, on conducting t-test analysis, the p-value obtained between RF and kNN was less than the threshold significance, which suggests that the k-NN performance was statistically different than RF.
Thus overall, these results once again imply that RF and kNN had similar performances averaged across 3 metrics for three problem sets, while LR performed poorly on all sets except one (ADULT).

Table 4: Training Set Performance over Metrics per Algorithm

MODEL	ACC	FSC	ROC	MEAN
LR	0.763	0.605	0.786	0.718
RF	**1.000**	**1.000**	**1.000**	**1.000**
KNN	*0.992	*0.982	*0.987	0.987

The format of Table 4 follows that of Table 2, where the classifiers are listed in different rows, and the different metrics form the different columns. Each entry in the table is the average ACC, FSC and ROC training score per classifier across all four datasets for five trials. (See Table 10 in Appendix for raw training scores per trial).
We observe that RF achieves perfect accuracy across all five trials, which is non-significant compared to the k-NN average training performance per metric for all five trials. (See p-values in Appendix Table 8). In other words, both RF and k-NN once again have similar training performances across all 4 problems across 5 trials for each metric. LR performs poorly in comparison with average training performance scores as well.

Table 5: Training Set Performance by Problem (averaged over three metrics)

MODEL	COV_TYPE	ADULT	LETTER.p1	LETTER.p2	MEAN
LR	0.756	0.782	0.608	0.733	0.792
RF	**1.000**	**1.000**	**1.000**	**1.000**	1.000
KNN	*1.000	*0.948	*1.000	*1.000	0.987

Table 5 follows the same format as Table 3, where the three rows list the different classifiers used, while the columns list the datasets used for comparison. Each entry in the table is the average training performance score across the three different metrics across 5 different trials.
We observe that RF achieves perfect training performance scores for all datasets across three metrics for all five trials, whereas k-NN achieves perfect training performance scores for three problem sets namely COV_TYPE, LETTER.p1 and LETTER.p2. (Since both classifiers had the same training performance score for three sets, I represented this as p-value 1.000 in Appendix Table 9).
For ADULT dataset, RF has the highest training performance score (1.000). This performance score was non-significant compared to k-NN training performance score ($p = 0.0893$). These results provide further evidence that RF and kNN had similar average performance scores across each problem set for five trials for the training set. LR had the worst training set average performance score across all problems for 5 trials.

Discussion

On comparing training and test results, RFs achieve consistently maximum performance scores (or non-significant from the best score) when evaluating the average performance for all problems per metric. (See Table 2). kNNs perform similarly across both training and test sets as well, with each score per metric non-statistically significant from RF performance (See Table 2 and Table 4). When analyzing per dataset, we observe that RF performs significantly better on COV_TYPE. LR, RF and k-NN perform similarly on ADULT. RF and kNN perform just as well on LETTER.p1, whereas k-NN performs slightly better on LETTER.p2.

In case of the training data, we hypothesize that both RF and kNN overfit our datasets to a certain extent. With RF, the min_sample_leaf parameter is set to its default value, i.e. 1. This means that the classifier will keep splitting feature attributes until there is one sample left at each leaf node. Since this tree is constructed on the training data, the tree overfits the training data and hence predicts on training data with 100% accuracy. This could be a reason why RFs have perfect training performance scores over five trials for all four problem sets and over three metrics. (See Table 4 and Table 5).
On the test data, RFs perform better than the other classifiers across three metric and across four problem sets, providing evidence that it is the best algorithm out of the three chosen for the study (See Table 3). This bolsters the CNM06 results that RFs had consistently better performances across problem sets and metrics than other classifiers.

In the case of kNN, we observe that this classifier has similar test performance scores as RF across problem sets and metrics. This memory-based classifier has a tendency to overfit the class distribution on unbalanced datasets (Argerich). The k-NN classifier used in the study has 26 neighbors, equally spaced (4-step values) from 1 to 101. For a lower number of neighbors, it is possible that k-NN overfits the training data, and thus has high training performance scores (See Table 4).
k-NN also tends to perform well on problem sets with fewer number of features, i.e. smaller input data. This is clearly displayed in Table 3 results, where k-NN performs optimally over all metrics for LETTER.p1 and LETTER.p2, as these problem sets have the least number of attributes. By contrast, k-NN performs poorly on COV_TYPE, the problem set that has the largest number of attributes. Thus, the lower dimensionality in certain problem sets could be another reason why RF and k-NN perform similarly in Table 2 and Table 3.

On the other hand, LR consistently performs significantly worse than RF and k-NN per metric and per problem across five trials as suggested by CNM06. As a classifier, LR is not very powerful as it

does not perform well on data that is not easily linearly separable. Real world data is noisy and complex, and the problem sets used in this paper reflect that fact. The ADULT dataset contains human data, whereas COV_TYPE contains data regarding vegetation cover, soil types and forest cover. The feature attributes of these problem sets are not entirely independent. This large amount of variance and non-linearly separable data could be a reason for the poor performance of LR across each problem set and metric.

Conclusion

Overall, the results of this study are consistent with those outlined in CNM06. RFs consistently perform well across four differently balanced datasets when evaluated using three different scoring metrics, as suggested by CNM06. While k-NNs have similar performance scores as RF, we hypothesize that this effect could be caused due to the overfitting nature of k-NNs as discussed in the Discussion section above. On the other hand, LR demonstrated the worst average performance on all metrics across all problem sets.

On exploring the different problem sets used in this analysis, we observe that ADULT has the lowest testing performance score out of all four problem sets. This could be due to the fact that it contains noisy human data, making it harder to predict on as compared to LETTER.p1, LETTER.p2 and COV_TYPE. Thus, while picking models for classification task, one must keep the No Free Lunch Theorem in mind, as no one model has optimal performance on all problem sets. Since this study was only conducted across a small subset of algorithms and metrics as compared to CNM06, certain variations in certain training and test results per problem set and classifier are expected. Despite these variances however, this study still provides enough evidence to indicate the robustness of random forests and the comparatively poor performance of Logistic Regression classifiers for the binary classification task, thus providing support to the replicability and reproducibility of the CNM06 study.

Bonus

In addition to the required 3 classifiers, Table 2 and Table 3 results, as well as the Appendix Tables, this paper includes Table 4: Training Set Performance over Metrics per Algorithm, Table 5: Training Set Performance by Problem (averaged over three metrics) as Main Matter Tables. Independent t-tests are conducted for each classifier based on training performance (similar to Table 6 and Table 7) and these p-value tables are included in the Appendix. (See Table 8 and Table 9). The Discussion section addresses the differences between training and test scores for each classifier and hypothesizes reasons for these performance scores. (Not sure if this counts, but I also made 2 t-test tables, one using t_test_rel and one using t_test_ind, included in the spreadsheets at the end of this project.)

Acknowledgements

I would like to thank and acknowledge the support of the COGS 118A teaching staff, Professor Fleischer and my classmates on Piazza for their guidance in understanding and completing this project. The homework notebooks as well as the Lecture 19 model selection notebook helped me format my code and structure my pipeline.

Appendix

Table 6: P-Vals for Table 2

MODEL	ACC	FSC	ROC
LR	1.28E-09	0.00029	0.003
RF	1.000	1.000	0.963
KNN	0.616	0.770	1.000

Table 7: P-Vals for Table 3

MODEL	COV_TYPE	ADULT	LETTER.p1	LETTER.p2
LR	1.75E-31	1.000	9.90E-05	2.44E-45
RF	1.000	0.738	0.139	1.98E-10
KNN	1.03E-22	0.220	1.000	1.000

Table 8: P-Vals for Table 4

MODEL	ACC	FSC	ROC
LR	2.23E-27	1.38E-09	2.91E-22
RF	1.000	1.000	1.000
KNN	0.324	0.324	0.324

Table 9: P-Vals for Table 5

MODEL	COV_TYPE	ADULT	LETTER.P1	LETTER.P2

LR	1.53E-44	1.69E-13	8.032E-06	6.32E-45
RF	1.000	1.000	1.000	1.000
KNN	1.000	0.0893	1.000	1.000

RAW TRIAL SCORES:

Table 10: Raw Training Performances by Trial

	TRIAL 1	TRIAL 2	TRIAL 3	TRIAL 4	TRIAL 5
COV_TYPE					
LR/ACC	0.760	0.749	0.759	0.753	0.754
LR/ROC	0.759	0.749	0.756	0.754	0.755
LR/FSC	0.766	0.747	0.763	0.757	0.756
RF/ACC	1.000	1.000	1.000	1.000	1.000
RF/ROC	1.000	1.000	1.000	1.000	1.000
RF/FSC	1.000	1.000	1.000	1.000	1.000
KNN/ACC	1.000	1.000	1.000	1.000	1.000
KNN/ROC	1.000	1.000	1.000	1.000	1.000
KNN/FSC	1.000	1.000	1.000	1.000	1.000
ADULT					
LR/ACC	0.822	0.826	0.820	0.824	0.808
LR/ROC	0.834	0.828	0.832	0.839	0.823
LR/FSC	0.699	0.696	0.699	0.699	0.682
RF/ACC	1.000	1.000	1.000	1.000	1.000
RF/ROC	1.000	1.000	1.000	1.000	1.000
RF/FSC	1.000	1.000	1.000	1.000	1.000
KNN/ACC	1.000	1.000	0.838	1.000	1.000
KNN/ROC	1.000	1.000	0.746	1.000	1.000

KNN/FSC	1.000	1.000	0.632	1.000	1.000
LETTER.p1					
LR/ACC	0.789	0.755	0.778	0.719	0.791
LR/ROC	0.852	0.795	0.831	0.806	0.851
LR/FSC	0.232	0.237	0.239	0.189	0.260
RF/ACC	1.000	1.000	1.000	1.000	1.000
RF/ROC	1.000	1.000	1.000	1.000	1.000
RF/FSC	1.000	1.000	1.000	1.000	1.000
KNN/ACC	1.000	1.000	1.000	1.000	1.000
KNN/ROC	1.000	1.000	1.000	1.000	1.000
KNN/FSC	1.000	1.000	1.000	1.000	1.000
LETTER.p2					
LR/ACC	0.737	0.733	0.725	0.729	0.738
LR/ROC	0.738	0.733	0.726	0.729	0.739
LR/FSC	0.738	0.738	0.724	0.734	0.740
RF/ACC	1.000	1.000	1.000	1.000	1.000
RF/ROC	1.000	1.000	1.000	1.000	1.000
RF/FSC	1.000	1.000	1.000	1.000	1.000
KNN/ACC	1.000	1.000	1.000	1.000	1.000
KNN/ROC	1.000	1.000	1.000	1.000	1.000
KNN/FSC	1.000	1.000	1.000	1.000	1.000

Table 11: Raw Testing Performances by Trial

	TRIAL 1	TRIAL 2	TRIAL 3	TRIAL 4	TRIAL 5
COV_TYPE					

LR/ACC	0.752	0.749	0.756	0.748	0.750
LR/ROC	0.753	0.750	0.755	0.749	0.751
LR/FSC	0.754	0.747	0.758	0.751	0.754
RF/ACC	0.824	0.819	0.825	0.818	0.822
RF/ROC	0.824	0.819	0.825	0.819	0.823
RF/FSC	0.825	0.817	0.823	0.819	0.822
KNN/ACC	0.778	0.776	0.787	0.775	0.777
KNN/ROC	0.778	0.775	0.787	0.775	0.777
KNN/FSC	0.776	0.771	0.785	0.778	0.779
ADULT					
LR/ACC	0.810	0.809	0.809	0.810	0.811
LR/ROC	0.819	0.821	0.818	0.820	0.821
LR/FSC	0.680	0.680	0.678	0.681	0.682
RF/ACC	0.852	0.854	0.849	0.849	0.852
RF/ROC	0.766	0.765	0.771	0.771	0.767
RF/FSC	0.662	0.663	0.665	0.664	0.662
KNN/ACC	0.838	0.838	0.837	0.836	0.835
KNN/ROC	0.742	0.744	0.744	0.742	0.737
KNN/FSC	0.623	0.626	0.624	0.624	0.616
LETTER.p1					
LR/ACC	0.791	0.742	0.780	0.719	0.795
LR/ROC	0.831	0.781	0.840	0.794	0.834
LR/FSC	0.244	0.234	0.234	0.192	0.239
RF/ACC	0.988	0.988	0.991	0.988	0.989
RF/ROC	0.856	0.862	0.898	0.864	0.878
RF/FSC	0.821	0.820	0.87	0.819	0.841
KNN/ACC	0.989	0.990	0.990	0.991	0.989

KNN/ROC	0.936	0.955	0.951	0.932	0.924
KNN/FSC	0.865	0.872	0.874	0.883	0.848
LETTER.p2					
LR/ACC	0.727	0.729	0.727	0.722	0.730
LR/ROC	0.727	0.729	0.727	0.722	0.730
LR/FSC	0.731	0.732	0.731	0.721	0.733
RF/ACC	0.946	0.944	0.948	0.947	0.945
RF/ROC	0.946	0.944	0.948	0.947	0.945
RF/FSC	0.946	0.944	0.948	0.946	0.945
KNN/ACC	0.956	0.949	0.954	0.953	0.952
KNN/ROC	0.956	0.949	0.954	0.953	0.952
KNN/FSC	0.956	0.949	0.953	0.953	0.952

References

Argerich L. "When should I use K-Nearest Neighbors? When is it a better algorithm than NN, SVM, etc.?" Quora. 20 Feb 2016, https://www.quora.com/When-should-I-use-K-Nearest-Neighbors-When-is-it-a-better-algorithm-than-NN-SVM-etc

Blackard J, Dean D, and Anderson C. “Covertype Data Set.” UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science. 1 Aug 1998, https://archive.ics.uci.edu/ml/datasets/covertype

Caruana, Rich, and Alexandru Niculescu-Mizil. “An Empirical Comparison of Supervised Learning Algorithms.” *Proceedings of the 23rd International Conference on Machine Learning - ICML '06*, 2006, doi:10.1145/1143844.1143865.

Koehrsen, Will. “Beyond Accuracy: Precision and Recall.” *Medium*, Towards Data Science, 10 Mar. 2018, towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c#:~:text=The%20metric%20our%20intuition%20tells,the%20number%20of%20false%20negatives.

Kohavi R, and Becker B. “Adult Data Set.” UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science. 1 May 1996, https://archive.ics.uci.edu/ml/datasets/adult

Narkhede, Sarang. “Understanding AUC - ROC Curve.” *Medium*, Towards Data Science, 14 Jan. 2021, towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

Slate D. “Letter Recognition Data Set.” UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science. 1 Jan 1991, https://archive.ics.uci.edu/ml/datasets/Letter+Recognition

In [138…]
```
import pandas as pd
import numpy as np
import string
import csv
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
from sklearn.model_selection import GridSearchCV
```

This notebook runs LOGREG over 4 problem sets across 5 trials. The values for Table 2 and Table 3 are recorded at each iteration of the loop.

Datasets

ADULT

In [92]:
```
ADULT_data = pd.read_csv('adult.data.csv', names = ['age', 'workclass', 'fnlw
                         'relationship', 'race', 'sex', 'capital-gain', 'capital
                         'native-country', 'target_income'
                        ])

ADULT_data['target_income'] = ADULT_data['target_income'].str.strip()

ADULT_data['target_income'] = ADULT_data.target_income.map( {'<=50K':0 , '>50

ADULT_one_hot_data = pd.get_dummies(ADULT_data,
                                    columns = ['workclass', 'education', 'mar
                                               'occupation', 'relationship', 'ra
                                    prefix = ['workclass', 'education', 'mari
                                               'occupation', 'relationship', 'ra

ADULT_one_hot_data = ADULT_one_hot_data.drop(['workclass_ ?',
                                              'occupation_ ?', 'native-countr

ADULT_one_hot_data[['age', 'fnlwgt', 'education-num', 'capital-gain',
    'capital-loss', 'hours-per-week']] = StandardScaler().fit_transform(ADULT_

ADULT_one_hot_data
```

Out[92]:

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	target_income	wor F
0	0.030671	-1.063611	1.134739	0.148453	-0.21666	-0.035429	0	
1	0.837109	-1.008707	1.134739	-0.145920	-0.21666	-2.222153	0	
2	-0.042642	0.245079	-0.420060	-0.145920	-0.21666	-0.035429	0	
3	1.057047	0.425801	-1.197459	-0.145920	-0.21666	-0.035429	0	
4	-0.775768	1.408176	1.134739	-0.145920	-0.21666	-0.035429	0	
…	…	…	…	…	…	…	…	
32556	-0.849080	0.639741	0.746039	-0.145920	-0.21666	-0.197409	0	
32557	0.103983	-0.335433	-0.420060	-0.145920	-0.21666	-0.035429	1	
32558	1.423610	-0.358777	-0.420060	-0.145920	-0.21666	-0.035429	0	

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	target_income	wor F
32559	-1.215643	0.110960	-0.420060	-0.145920	-0.21666	-1.655225	0	
32560	0.983734	0.929893	-0.420060	1.888424	-0.21666	-0.035429	1	

32561 rows × 106 columns

Balance of dataset

In [93]:
```
positive_labels = ADULT_data['target_income'].value_counts()[1]/ ADULT_data['
negative_labels = ADULT_data['target_income'].value_counts()[0]/ ADULT_data['
print("% of negative labels:", negative_labels)
print("% of positive labels:", positive_labels)
```

```
% of negative labels: 75.91904425539757
% of positive labels: 24.080955744602438
```

COV_type data

In [94]:
```
COV_type_data = pd.read_csv('covtype.data.gz', header = None)

cols = [c for c in COV_type_data.columns]
cols[-1] = 'forest_cover'
COV_type_data.columns = cols



largest_class = COV_type_data['forest_cover'].value_counts().idxmax()

COV_type_data.loc[COV_type_data['forest_cover'] != largest_class, 'forest_cov
COV_type_data.loc[COV_type_data['forest_cover'] == largest_class, 'forest_cov



COV_type_data.iloc[:, :-1] = StandardScaler().fit_transform(COV_type_data.ilo

COV_type_data
```

Out[94]:

	0	1	2	3	4	5	6	
0	-1.297805	-0.935157	-1.482820	-0.053767	-0.796273	-1.180146	0.330743	0.4391
1	-1.319235	-0.890480	-1.616363	-0.270188	-0.899197	-1.257106	0.293388	0.5908
2	-0.554907	-0.148836	-0.681563	-0.006719	0.318742	0.532212	0.816364	0.7426
3	-0.622768	-0.005869	0.520322	-0.129044	1.227908	0.474492	0.965786	0.7426
4	-1.301377	-0.988770	-1.616363	-0.547771	-0.813427	-1.256464	0.293388	0.5403
...	...	...	...	...	...	...	...	
581007	-2.012130	-0.023740	0.787408	-0.867697	-0.504653	-1.437962	1.040496	0.6920
581008	-2.029988	-0.032675	0.653865	-0.952383	-0.590424	-1.446299	1.040496	0.6920
581009	-2.047847	0.029873	0.386780	-0.985317	-0.676194	-1.449506	0.891075	0.8944
581010	-2.054990	0.128163	0.119694	-0.985317	-0.710502	-1.449506	0.666942	1.0967
581011	-2.058562	0.083486	-0.147392	-0.985317	-0.727656	-1.464256	0.704298	1.0461

581012 rows × 55 columns

Balance of dataset

Treat largest class as positive class. The rest are negative.

In [77]:
```
# positive_labels = len(COV_type_data[COV_type_data['Forest cover'] == 7])/le

positive_labels = COV_type_data['forest_cover'].value_counts().max()/len(COV_
negative_labels = len(COV_type_data[COV_type_data['forest_cover'] != COV_type_

print("% of negative labels:", negative_labels)
print("% of positive labels:", positive_labels)
```

```
% of negative labels: 48.75992234239568
% of positive labels: 51.240077657604324
```

LETTER

In [78]:
```
LETTER_p1 = pd.read_csv('letter-recognition.data', header = None)

cols = [c for c in LETTER_p1.columns]
cols[0] = 'letter'
LETTER_p1.columns = cols

# LETTER_p1["letter"] = LETTER_p1["letter"].astype(str).astype(int)


LETTER_p1.loc[:, LETTER_p1.columns != 'letter'] = StandardScaler().fit_transf
LETTER_p1
```

Out[78]:

	letter	1	2	3	4	5	6	7
0	T	-1.057698	0.291877	-1.053277	-0.164704	-1.144013	0.544130	2.365097
1	I	0.510385	1.502358	-1.053277	0.719730	-0.687476	1.531305	-1.075326
2	D	-0.012309	1.199738	0.435910	1.161947	1.138672	1.531305	-0.645273
3	N	1.555774	1.199738	0.435910	0.277513	-0.230939	-0.936631	0.644886
4	G	-1.057698	-1.826464	-1.053277	-1.933571	-1.144013	0.544130	-0.645273
...	...	...	...	...	...	...	...	...
19995	D	-1.057698	-1.523844	-1.053277	-1.049137	-0.687476	0.050543	-0.215220
19996	C	1.555774	0.897117	1.428701	1.161947	0.225598	-1.430218	0.214833
19997	T	1.033079	0.594497	0.435910	0.719730	0.682135	-0.443044	1.504991
19998	S	-1.057698	-1.221224	-0.556881	-1.491354	-1.144013	0.544130	-0.215220
19999	A	-0.012309	0.594497	0.435910	0.277513	-0.687476	1.037718	-1.075326

20000 rows × 17 columns

Letter.p1 - treat O as positive class, rest as negative

Unbalanced dataset

In [79]:
```
O_list = ['O']

# LETTER_p1['letter'] = pd.to_numeric(LETTER_p1['letter'])
```

```
LETTER_p1.loc[~LETTER_p1['letter'].isin(O_list), 'letter'] = 0
LETTER_p1.loc[LETTER_p1['letter'].isin(O_list), 'letter'] = 1

LETTER_p1['letter'].value_counts()
```

Out[79]:
```
0    19247
1      753
Name: letter, dtype: int64
```

In [80]:
```
positive_labels = len(LETTER_p1[LETTER_p1['letter'] == 1])/len(LETTER_p1['let
negative_labels = len(LETTER_p1[LETTER_p1['letter'] == 0])/len(LETTER_p1['let

print("% of negative labels:", negative_labels)
print("% of positive labels:", positive_labels)
```

```
% of negative labels: 96.235
% of positive labels: 3.765
```

Letter.p2 - treat A-M as positive class, rest as negative

In [81]:
```
LETTER_p2 = pd.read_csv('letter-recognition.data', header = None)

cols = [c for c in LETTER_p2.columns]
cols[0] = 'letter'
LETTER_p2.columns = cols


LETTER_p2.loc[:, LETTER_p2.columns != 'letter'] = StandardScaler().fit_transf

pos_alphabet_list = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', '
neg_alphabet_list = sorted(list(set(string.ascii_uppercase) - set(pos_alphabe




LETTER_p2.loc[LETTER_p2['letter'].isin(pos_alphabet_list), 'letter'] = 1
LETTER_p2.loc[LETTER_p2['letter'].isin(neg_alphabet_list), 'letter'] = 0

LETTER_p1["letter"] = LETTER_p1["letter"].astype(str).astype(int)
LETTER_p2["letter"] = LETTER_p2["letter"].astype(str).astype(int)

LETTER_p2
```

Out[81]:

	letter	1	2	3	4	5	6	7
0	0	-1.057698	0.291877	-1.053277	-0.164704	-1.144013	0.544130	2.365097
1	1	0.510385	1.502358	-1.053277	0.719730	-0.687476	1.531305	-1.075326
2	1	-0.012309	1.199738	0.435910	1.161947	1.138672	1.531305	-0.645273
3	0	1.555774	1.199738	0.435910	0.277513	-0.230939	-0.936631	0.644886
4	1	-1.057698	-1.826464	-1.053277	-1.933571	-1.144013	0.544130	-0.645273
...	...	...	...	...	...	...	...	...
19995	1	-1.057698	-1.523844	-1.053277	-1.049137	-0.687476	0.050543	-0.215220
19996	1	1.555774	0.897117	1.428701	1.161947	0.225598	-1.430218	0.214833
19997	0	1.033079	0.594497	0.435910	0.719730	0.682135	-0.443044	1.504991
19998	0	-1.057698	-1.221224	-0.556881	-1.491354	-1.144013	0.544130	-0.215220
19999	1	-0.012309	0.594497	0.435910	0.277513	-0.687476	1.037718	-1.075326

20000 rows × 17 columns

In [82]:
```
LETTER_p1['letter']
```

Out[82]:
```
0        0
1        0
2        0
3        0
4        0
        ..
19995    0
19996    0
19997    0
19998    0
19999    0
Name: letter, Length: 20000, dtype: int64
```

Well-balanced dataset

In [83]:
```
pos_alphabet_list = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L
neg_alphabet_list = sorted(list(set(string.ascii_uppercase) - set(pos_alphabet


positive_labels = len(LETTER_p2[LETTER_p2['letter'] == 1])/len(LETTER_p2['let
negative_labels = len(LETTER_p2[LETTER_p2['letter'] == 0])/len(LETTER_p2['let

print("% of negative labels:", negative_labels)
print("% of positive labels:", positive_labels)
```

```
% of negative labels: 50.3
% of positive labels: 49.7
```

Experiment - LogisticRegression() over 4 datasets over 5 trials

In [84]:
```
def split_data(data, column):

    Y = data[column]
    X = data.drop([column], axis=1)

    X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size=5000

    return X_train, X_test, y_train, y_test
```

In [85]:
```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score, accuracy_score, roc_auc_score
from sklearn.model_selection import train_test_split
```

In [131…
```
accuracy_metric = []
f1_score_metric = []
roc_auc_score_metric = []

ADULT_metric = []
COV_type_metric = []
LETTER_p1_metric = []
LETTER_p2_metric = []
```

```
datalist = [COV_type_data, ADULT_one_hot_data, LETTER_p1, LETTER_p2]


for ind, data in enumerate(datalist):
    for i in range(5):

        print('Start of trial', i+1)

        # COV_type data

        if ind == 0:
            print('At COV_type_data')
            dataset = 'COV_type_data'
            X_train, X_test, y_train, y_test = split_data(data, 'forest_cover


        # ADULT_data

        elif ind == 1:
            print('At ADULT_data')
            dataset = 'ADULT_data'
            X_train, X_test, y_train, y_test = split_data(data, 'target_incom


        # LETTER.p1 data

        if ind == 2:
            print('At LETTER_p1')
            dataset = 'LETTER_p1'
            X_train, X_test, y_train, y_test = split_data(data, 'letter')


        # LETTER.p2 data
        if ind == 3:
            print('At LETTER_p2')
            dataset = 'LETTER_p2'
            X_train, X_test, y_train, y_test = split_data(data, 'letter')




        pipe = Pipeline([('classifier', LogisticRegression())
                         ])



        search_space = [
                {'classifier': [LogisticRegression(class_weight = 'balanced',
                 'classifier__solver': ['saga'],
                 'classifier__penalty': ['l1', 'l2'],
                 'classifier__C': [0.00000001, 0.0000001, 0.000001, 0.0001, 0
                {'classifier': [LogisticRegression(class_weight = 'balanced',
                 'classifier__solver': ['sag'],
                 'classifier__penalty': ['l2'],
                 'classifier__C': [0.00000001, 0.0000001, 0.000001, 0.0001, 0
                {'classifier': [LogisticRegression(class_weight = 'balanced',
                 'classifier__solver': ['sag','saga'],
                 'classifier__penalty': ['none']}
                ]


        # Create grid search
        clf = GridSearchCV(pipe, search_space, cv=StratifiedKFold(n_splits=5)
                   scoring=['accuracy','roc_auc', 'f1'], refit='accuracy',
                   verbose=0, n_jobs = -1)
```

```
        # Fit grid search
        best_model = clf.fit(X_train, y_train)

        # Get best hyperparameters for accuracy, roc_auc score, f1_score

        best_acc_param = best_model.cv_results_['params'][ np.argmin(best_mod

        best_auc_param = best_model.cv_results_['params'][np.argmin(best_mode
        best_f1_param = best_model.cv_results_['params'][np.argmin(best_model


        # Train 3 models using the 5000 samples and each of the 3 best parame

        # Tuned for accuracy
        acc_model = best_acc_param['classifier'].fit(X_train, y_train)

        # Tuned for roc-auc score
        auc_model = best_auc_param['classifier'].fit(X_train, y_train)

        # Tuned for f1 score
        f1_model =  best_f1_param['classifier'].fit(X_train, y_train)



        # fit a classifier using that best param on the training set,
        # predict the training set, and record the corresponding training set

       # Predict on training data

        y_pred_acc_tr = acc_model.predict(X_train)
        y_pred_auc_tr = auc_model.predict(X_train)
        y_pred_f1_tr = f1_model.predict(X_train)



        print('Trial ', i+1, ' raw training scores for', dataset)

         # Raw train accuracy score
        print(accuracy_score(y_train, y_pred_acc_tr))

         # Raw train roc_auc score
        print(roc_auc_score(y_train, y_pred_auc_tr))

         # Raw train f1_score
        print(f1_score(y_train, y_pred_f1_tr))



        # Predict on Test data

        y_pred_acc = acc_model.predict(X_test)
        y_pred_auc = auc_model.predict(X_test)
        y_pred_f1 = f1_model.predict(X_test)


        print('Trial ', i+1, ' raw test scores for', dataset)
        # Raw test accuracy score
        print(accuracy_score(y_test, y_pred_acc))

        # Raw test roc_auc score
        print(roc_auc_score(y_test, y_pred_auc))

        # Raw test f1_score
```

```
        print(f1_score(y_test, y_pred_f1))

        # Append raw test scores to list to generate Table 2 values

        accuracy_metric.append(accuracy_score(y_test, y_pred_acc))
        roc_auc_score_metric.append(roc_auc_score(y_test, y_pred_auc))
        f1_score_metric.append(f1_score(y_test, y_pred_f1))

        # For Table 3
        if ind == 0:
            COV_type_metric.extend([accuracy_score(y_test, y_pred_acc), roc_a
                                    f1_score(y_test, y_pred_f1)])
        elif ind == 1:
            ADULT_metric.extend([accuracy_score(y_test, y_pred_acc), roc_auc_
                                    f1_score(y_test, y_pred_f1)])
        elif ind == 2:
            LETTER_p1_metric.extend([accuracy_score(y_test, y_pred_acc), roc_
                                    f1_score(y_test, y_pred_f1)])

        elif ind == 3:
            LETTER_p2_metric.extend([accuracy_score(y_test, y_pred_acc), roc_
                                    f1_score(y_test, y_pred_f1)])


        print("End of Trial", i+1)
        print('-----------------------------------------')
        print()
```

```
Start of trial 1
At COV_type_data
Trial  1  raw training scores for COV_type_data
0.7596
0.7593637483797667
0.7663297045101088
Trial  1  raw test scores for COV_type_data
0.7520815538565169
0.7527078075359809
0.753565265410368
End of Trial 1
-----------------------------------------

Start of trial 2
At COV_type_data
Trial  2  raw training scores for COV_type_data
0.7486
0.7493004333972704
0.7468277945619335
Trial  2  raw test scores for COV_type_data
0.7491805726269591
0.7498516542587442
0.7513274869402825
End of Trial 2
-----------------------------------------

Start of trial 3
At COV_type_data
Trial  3  raw training scores for COV_type_data
0.759
0.7559940641803302
0.7630285152409046
Trial  3  raw test scores for COV_type_data
0.7561977875460928
0.7549119095581273
0.758020577202417
```

```
End of Trial 3
----------------------------------------

Start of trial 4
At COV_type_data
Trial  4  raw training scores for COV_type_data
0.7532
0.7535900263593738
0.7567034700315458
Trial  4  raw test scores for COV_type_data
0.7477847683728811
0.7485723195840006
0.7513057053890412
End of Trial 4
----------------------------------------

Start of trial 5
At COV_type_data
Trial  5  raw training scores for COV_type_data
0.7538
0.7545462870741435
0.7563823471205224
Trial  5  raw test scores for COV_type_data
0.7496093831378513
0.7505187313944549
0.7540508923729651
End of Trial 5
----------------------------------------

Start of trial 1
At ADULT_data
Trial  1  raw training scores for ADULT_data
0.822
0.8342134707907394
0.698509485094851
Trial  1  raw test scores for ADULT_data
0.8101665396756286
0.8194881679820121
0.6800391389432485
End of Trial 1
----------------------------------------

Start of trial 2
At ADULT_data
Trial  2  raw training scores for ADULT_data
0.8258
0.8278707743694388
0.6961981164980815
Trial  2  raw test scores for ADULT_data
0.8086426472188962
0.8210035123406253
0.6800145613396432
End of Trial 2
----------------------------------------

Start of trial 3
At ADULT_data
Trial  3  raw training scores for ADULT_data
0.82
0.8321016566918206
0.6987951807228915
Trial  3  raw test scores for ADULT_data
0.8094045934472625
0.818024896039302
0.6778288868445262
End of Trial 3
----------------------------------------

Start of trial 4
```

```
At ADULT_data
Trial  4  raw training scores for ADULT_data
0.8236
0.8391956612498804
0.6993865030674846
Trial  4  raw test scores for ADULT_data
0.810239105983092
0.8197948307141647
0.6810197609172969
End of Trial 4
----------------------------------------

Start of trial 5
At ADULT_data
Trial  5  raw training scores for ADULT_data
0.8076
0.8232974988551865
0.6822985468956407
Trial  5  raw test scores for ADULT_data
0.8107833532890679
0.8213280935950479
0.681565610307138
End of Trial 5
----------------------------------------

Start of trial 1
At LETTER_p1
Trial  1  raw training scores for LETTER_p1
0.7892
0.8518102053597838
0.23177842565597662
Trial  1  raw test scores for LETTER_p1
0.7905333333333333
0.831481658615907
0.24434824434824431
End of Trial 1
----------------------------------------

Start of trial 2
At LETTER_p1
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/_sag.py:329: Conve
rgenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/_sag.py:329: Conve
rgenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
Trial  2  raw training scores for LETTER_p1
0.7546
0.7951705855232827
0.23682310469314075
Trial  2  raw test scores for LETTER_p1
0.7423333333333333
0.7805795706639347
0.2341623650868137
End of Trial 2
----------------------------------------

Start of trial 3
At LETTER_p1
Trial  3  raw training scores for LETTER_p1
0.7778
0.8305345035599585
0.23851953392734748
Trial  3  raw test scores for LETTER_p1
0.7798
0.8399124472590886
0.2338204592901879
End of Trial 3
----------------------------------------
```

```
Start of trial 4
At LETTER_p1
Trial  4  raw training scores for LETTER_p1
0.7186
0.8061592063643107
0.1881130986728217
Trial  4  raw test scores for LETTER_p1
0.7190666666666666
0.7943620698680305
0.19210122699386503
End of Trial 4
-----------------------------------------

Start of trial 5
At LETTER_p1
Trial  5  raw training scores for LETTER_p1
0.7912
0.8507291666666666
0.25957446808510637
Trial  5  raw test scores for LETTER_p1
0.795
0.8339928911450484
0.23904974016332595
End of Trial 5
-----------------------------------------

Start of trial 1
At LETTER_p2
Trial  1  raw training scores for LETTER_p2
0.7374
0.7377399758231402
0.7376623376623376
Trial  1  raw test scores for LETTER_p2
0.7274
0.7274171120834698
0.7314991135333904
End of Trial 1
-----------------------------------------

Start of trial 2
At LETTER_p2
Trial  2  raw training scores for LETTER_p2
0.7334
0.7334061173449787
0.7373399014778325
Trial  2  raw test scores for LETTER_p2
0.729
0.729119067795111
0.7315236774321379
End of Trial 2
-----------------------------------------

Start of trial 3
At LETTER_p2
Trial  3  raw training scores for LETTER_p2
0.7254
0.7255578688054413
0.7244631747943008
Trial  3  raw test scores for LETTER_p2
0.7268
0.7272405719748699
0.7314899751015594
End of Trial 3
-----------------------------------------

Start of trial 4
At LETTER_p2
Trial  4  raw training scores for LETTER_p2
```

```
0.7292
0.7291058627752176
0.7338836477987422
Trial  4  raw test scores for LETTER_p2
0.7218666666666667
0.7219118264122495
0.720899116938721
End of Trial 4
-----------------------------------------

Start of trial 5
At LETTER_p2
Trial  5  raw training scores for LETTER_p2
0.7384
0.7385597005933118
0.7397532829287704
Trial  5  raw test scores for LETTER_p2
0.7300666666666666
0.7302656584052929
0.7334956888040545
End of Trial 5
-----------------------------------------
```

For Table 2

In [132…

```
print('Accuracy metric values across all datasets, across 5 trials, for LR:')
print(accuracy_metric)

print()

print()
print('F-score metric values across all datasets, across 5 trials, for LR:')
print(f1_score_metric)

print()

print('ROC_AUC metric values across all datasets, across 5 trials, for LR:')
print(roc_auc_score_metric)

print()

print('Average scores for each metric: ')
print('ACC:', sum(accuracy_metric)/len(accuracy_metric))
print('FSC:', sum(f1_score_metric)/len(f1_score_metric))
print('ROC_AUC:', sum(roc_auc_score_metric)/len(roc_auc_score_metric))
```

```
Accuracy metric values across all datasets, across 5 trials, for LR:
[0.7520815538565169, 0.7491805726269591, 0.7561977875460928, 0.747784768372881
1, 0.7496093831378513, 0.8101665396756286, 0.8086426472188962, 0.8094045934472
625, 0.810239105983092, 0.8107833532890679, 0.7905333333333333, 0.742333333333
3333, 0.7798, 0.7190666666666666, 0.795, 0.7274, 0.729, 0.7268, 0.721866666666
6667, 0.7300666666666666]


F-score metric values across all datasets, across 5 trials, for LR:
[0.753565265410368, 0.7513274869402825, 0.758020577202417, 0.7513057053890412,
0.7540508923729651, 0.6800391389432485, 0.6800145613396432, 0.677828886844526
2, 0.6810197609172969, 0.681565610307138, 0.24434824434824431, 0.2341623650868
137, 0.2338204592901879, 0.19210122699386503, 0.23904974016332595, 0.731499113
5333904, 0.7315236774321379, 0.7314899751015594, 0.720899116938721, 0.73349568
88040545]

ROC_AUC metric values across all datasets, across 5 trials, for LR:
[0.7527078075359809, 0.7498516542587442, 0.7549119095581273, 0.748572319584000
6, 0.7505187313944549, 0.8194881679820121, 0.8210035123406253, 0.8180248960393
02, 0.8197948307141647, 0.8213280935950479, 0.831481658615907, 0.7805795706639
347, 0.8399124472590886, 0.7943620698680305, 0.8339928911450484, 0.72741711208
```

```
34698, 0.729119067795111, 0.7272405719748699, 0.7219118264122495, 0.7302656584
052929]

Average scores for each metric:
ACC: 0.7632978485910457
FSC: 0.5980563746679615
ROC_AUC: 0.778624239861273
```

Write the 20 values for each metric to a .csv to do p-test comparisons.

In [140…
```
with open('Table_2_p_test', 'w') as f:

    # using csv.writer method from CSV package
    write = csv.writer(f)

    write.writerow(accuracy_metric)

    write.writerow(f1_score_metric)

    write.writerow(roc_auc_score_metric)
```

for Table 3

In [133…
```
print('COV_type')
print('Metric values across 5 trials, for LR:')
print(COV_type_metric)

print()

print('ADULT')
print('Metric values across 5 trials, for LR:')
print(ADULT_metric)

print()

print('LETTER.p1')
print('Metric values across 5 trials, for LR:')
print(LETTER_p1_metric)

print()

print('LETTER.p2')
print('Metric values across 5 trials, for LR:')
print(LETTER_p2_metric)
```

```
COV_type
Metric values across 5 trials, for LR:
[0.7520815538565169, 0.7527078075359809, 0.753565265410368, 0.749180572626959
1, 0.7498516542587442, 0.7513274869402825, 0.7561977875460928, 0.7549119095581
273, 0.758020577202417, 0.7477847683728811, 0.7485723195840006, 0.751305705389
0412, 0.7496093831378513, 0.7505187313944549, 0.7540508923729651]

ADULT
Metric values across 5 trials, for LR:
[0.8101665396756286, 0.8194881679820121, 0.6800391389432485, 0.808642647218896
2, 0.8210035123406253, 0.6800145613396432, 0.8094045934472625, 0.8180248960393
02, 0.6778288868445262, 0.810239105983092, 0.8197948307141647, 0.6810197609172
969, 0.8107833532890679, 0.8213280935950479, 0.681565610307138]

LETTER.p1
Metric values across 5 trials, for LR:
[0.7905333333333333, 0.831481658615907, 0.24434824434824431, 0.742333333333333
3, 0.7805795706639347, 0.2341623650868137, 0.7798, 0.8399124472590886, 0.23382
04592901879, 0.7190666666666666, 0.7943620698680305, 0.19210122699386503, 0.79
5, 0.8339928911450484, 0.23904974016332595]
```

```
LETTER.p2
Metric values across 5 trials, for LR:
[0.7274, 0.7274171120834698, 0.7314991135333904, 0.729, 0.729119067795111, 0.7
315236774321379, 0.7268, 0.7272405719748699, 0.7314899751015594, 0.72186666666
66667, 0.7219118264122495, 0.720899116938721, 0.7300666666666666, 0.7302656584
052929, 0.7334956888040545]
```

Write the 15 values for each metric to a .csv to do p-test comparisons.

In [142…
```
with open('Table_3_p_test', 'w') as f:

    # using csv.writer method from CSV package
    write = csv.writer(f)

    write.writerow(COV_type_metric)

    write.writerow(ADULT_metric)

    write.writerow(LETTER_p1_metric)

    write.writerow(LETTER_p2_metric)
```

In [135…
```
print('Average metric scores for each dataset across 5 trials: ')

print()
print('COV_type:', sum(COV_type_metric)/len(COV_type_metric))
print()

print('ADULT:', sum(ADULT_metric)/len(ADULT_metric))
print()

print('LETTER.p1:', sum(LETTER_p1_metric)/len(LETTER_p1_metric))
print()

print('LETTER.p2:', sum(LETTER_p2_metric)/len(LETTER_p2_metric))
```

```
Average metric scores for each dataset across 5 trials:

COV_type: 0.7519790943457789

ADULT: 0.7699562465757966

LETTER.p1: 0.6033696004511852

LETTER.p2: 0.727999676120946
```

In [41]:
```
import pandas as pd
import numpy as np
import string
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
import csv
from sklearn.svm import SVC
```

This notebook runs RF across 4 problem sets across 5 trials. The values for Table 2 and Table 3 are recorded at each iteration of the loop.

Datasets

ADULT

In [34]:
```
ADULT_data = pd.read_csv('adult.data.csv', names = ['age', 'workclass', 'fnlw
                         'relationship', 'race', 'sex', 'capital-gain', 'capital
                         'native-country', 'target_income'
                        ])

ADULT_data['target_income'] = ADULT_data['target_income'].str.strip()

ADULT_data['target_income'] = ADULT_data.target_income.map( {'<=50K':0 , '>50

ADULT_one_hot_data = pd.get_dummies(ADULT_data,
                                    columns = ['workclass', 'education', 'mar
                                               'occupation', 'relationship', 'ra
                                    prefix = ['workclass', 'education', 'mari
                                               'occupation', 'relationship', 'ra

ADULT_one_hot_data = ADULT_one_hot_data.drop(['workclass_ ?',
                                              'occupation_ ?', 'native-countr

ADULT_one_hot_data[['age', 'fnlwgt', 'education-num', 'capital-gain',
   'capital-loss', 'hours-per-week']] = StandardScaler().fit_transform(ADULT_

ADULT_one_hot_data
```

Out[34]:

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	target_income	wor F
0	0.030671	-1.063611	1.134739	0.148453	-0.21666	-0.035429	0	
1	0.837109	-1.008707	1.134739	-0.145920	-0.21666	-2.222153	0	
2	-0.042642	0.245079	-0.420060	-0.145920	-0.21666	-0.035429	0	
3	1.057047	0.425801	-1.197459	-0.145920	-0.21666	-0.035429	0	
4	-0.775768	1.408176	1.134739	-0.145920	-0.21666	-0.035429	0	
...	...	...	...	...	...	...	...	
32556	-0.849080	0.639741	0.746039	-0.145920	-0.21666	-0.197409	0	
32557	0.103983	-0.335433	-0.420060	-0.145920	-0.21666	-0.035429	1	
32558	1.423610	-0.358777	-0.420060	-0.145920	-0.21666	-0.035429	0	
32559	-1.215643	0.110960	-0.420060	-0.145920	-0.21666	-1.655225	0	

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	target_income	wor F
32560	0.983734	0.929893	-0.420060	1.888424	-0.21666	-0.035429	1	

32561 rows × 106 columns

Balance of dataset

In [32]:
```
positive_labels = ADULT_data['target_income'].value_counts()[1]/ ADULT_data['
negative_labels = ADULT_data['target_income'].value_counts()[0]/ ADULT_data['
print("% of negative labels:", negative_labels)
print("% of positive labels:", positive_labels)
```

```
% of negative labels: 75.91904425539757
% of positive labels: 24.080955744602438
```

Unbalanced dataset

COV_type data

In [35]:
```
COV_type_data = pd.read_csv('covtype.data.gz', header = None)

cols = [c for c in COV_type_data.columns]
cols[-1] = 'forest_cover'
COV_type_data.columns = cols



largest_class = COV_type_data['forest_cover'].value_counts().idxmax()

COV_type_data.loc[COV_type_data['forest_cover'] != largest_class, 'forest_cov
COV_type_data.loc[COV_type_data['forest_cover'] == largest_class, 'forest_cov



COV_type_data.iloc[:, :-1] = StandardScaler().fit_transform(COV_type_data.ilo

COV_type_data
```

Out[35]:

	0	1	2	3	4	5	6	
0	-1.297805	-0.935157	-1.482820	-0.053767	-0.796273	-1.180146	0.330743	0.4391
1	-1.319235	-0.890480	-1.616363	-0.270188	-0.899197	-1.257106	0.293388	0.5908
2	-0.554907	-0.148836	-0.681563	-0.006719	0.318742	0.532212	0.816364	0.7426
3	-0.622768	-0.005869	0.520322	-0.129044	1.227908	0.474492	0.965786	0.7426
4	-1.301377	-0.988770	-1.616363	-0.547771	-0.813427	-1.256464	0.293388	0.5403
...	...	...	...	...	...	...	...	
581007	-2.012130	-0.023740	0.787408	-0.867697	-0.504653	-1.437962	1.040496	0.6920
581008	-2.029988	-0.032675	0.653865	-0.952383	-0.590424	-1.446299	1.040496	0.6920
581009	-2.047847	0.029873	0.386780	-0.985317	-0.676194	-1.449506	0.891075	0.8944
581010	-2.054990	0.128163	0.119694	-0.985317	-0.710502	-1.449506	0.666942	1.0967
581011	-2.058562	0.083486	-0.147392	-0.985317	-0.727656	-1.464256	0.704298	1.0461

581012 rows × 55 columns

Balance of dataset

Treat largest class as positive class. The rest are negative.

In [37]:
```
# positive_labels = len(COV_type_data[COV_type_data['Forest cover'] == 7])/le

positive_labels = COV_type_data['forest_cover'].value_counts().max()/len(COV_
negative_labels = len(COV_type_data[COV_type_data['forest_cover'] != COV_type

print("% of negative labels:", negative_labels)
print("% of positive labels:", positive_labels)
```

```
% of negative labels: 48.75992234239568
% of positive labels: 51.240077657604324
```

LETTER

In [5]:
```
LETTER_p1 = pd.read_csv('letter-recognition.data', header = None)

cols = [c for c in LETTER_p1.columns]
cols[0] = 'letter'
LETTER_p1.columns = cols



LETTER_p1.loc[:, LETTER_p1.columns != 'letter'] = StandardScaler().fit_transf
LETTER_p1
```

Out[5]:

	letter	1	2	3	4	5	6	7
0	T	-1.057698	0.291877	-1.053277	-0.164704	-1.144013	0.544130	2.365097
1	I	0.510385	1.502358	-1.053277	0.719730	-0.687476	1.531305	-1.075326
2	D	-0.012309	1.199738	0.435910	1.161947	1.138672	1.531305	-0.645273
3	N	1.555774	1.199738	0.435910	0.277513	-0.230939	-0.936631	0.644886
4	G	-1.057698	-1.826464	-1.053277	-1.933571	-1.144013	0.544130	-0.645273
...	...	...	...	...	...	...	...	...
19995	D	-1.057698	-1.523844	-1.053277	-1.049137	-0.687476	0.050543	-0.215220
19996	C	1.555774	0.897117	1.428701	1.161947	0.225598	-1.430218	0.214833
19997	T	1.033079	0.594497	0.435910	0.719730	0.682135	-0.443044	1.504991
19998	S	-1.057698	-1.221224	-0.556881	-1.491354	-1.144013	0.544130	-0.215220
19999	A	-0.012309	0.594497	0.435910	0.277513	-0.687476	1.037718	-1.075326

20000 rows × 17 columns

Letter.p1 - treat O as positive class, rest as negative

Unbalanced dataset

In [6]:
```
O_list = ['O']

# LETTER_p1['letter'] = pd.to_numeric(LETTER_p1['letter'])
```

```
LETTER_p1.loc[~LETTER_p1['letter'].isin(O_list), 'letter'] = 0
LETTER_p1.loc[LETTER_p1['letter'].isin(O_list), 'letter'] = 1

LETTER_p1['letter'].value_counts()
```

Out[6]:
```
0    19247
1      753
Name: letter, dtype: int64
```

In [7]:
```
positive_labels = len(LETTER_p1[LETTER_p1['letter'] == 1])/len(LETTER_p1['let
negative_labels = len(LETTER_p1[LETTER_p1['letter'] == 0])/len(LETTER_p1['let

print("% of negative labels:", negative_labels)
print("% of positive labels:", positive_labels)
```

```
% of negative labels: 96.235
% of positive labels: 3.765
```

Letter.p2 - treat A-M as positive class, rest as negative

In [10]:
```
LETTER_p2 = pd.read_csv('letter-recognition.data', header = None)

cols = [c for c in LETTER_p2.columns]
cols[0] = 'letter'
LETTER_p2.columns = cols


LETTER_p2.loc[:, LETTER_p2.columns != 'letter'] = StandardScaler().fit_transf

pos_alphabet_list = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', '
neg_alphabet_list = sorted(list(set(string.ascii_uppercase) - set(pos_alphabe




LETTER_p2.loc[LETTER_p2['letter'].isin(pos_alphabet_list), 'letter'] = 1
LETTER_p2.loc[LETTER_p2['letter'].isin(neg_alphabet_list), 'letter'] = 0

LETTER_p1["letter"] = LETTER_p1["letter"].astype(str).astype(int)
LETTER_p2["letter"] = LETTER_p2["letter"].astype(str).astype(int)

LETTER_p2
```

Out[10]:

	letter	1	2	3	4	5	6	7
0	0	-1.057698	0.291877	-1.053277	-0.164704	-1.144013	0.544130	2.365097
1	1	0.510385	1.502358	-1.053277	0.719730	-0.687476	1.531305	-1.075326
2	1	-0.012309	1.199738	0.435910	1.161947	1.138672	1.531305	-0.645273
3	0	1.555774	1.199738	0.435910	0.277513	-0.230939	-0.936631	0.644886
4	1	-1.057698	-1.826464	-1.053277	-1.933571	-1.144013	0.544130	-0.645273
...	...	...	...	...	...	...	...	...
19995	1	-1.057698	-1.523844	-1.053277	-1.049137	-0.687476	0.050543	-0.215220
19996	1	1.555774	0.897117	1.428701	1.161947	0.225598	-1.430218	0.214833
19997	0	1.033079	0.594497	0.435910	0.719730	0.682135	-0.443044	1.504991
19998	0	-1.057698	-1.221224	-0.556881	-1.491354	-1.144013	0.544130	-0.215220
19999	1	-0.012309	0.594497	0.435910	0.277513	-0.687476	1.037718	-1.075326

20000 rows × 17 columns

Well-balanced dataset

In [12]:
```
pos_alphabet_list = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L
neg_alphabet_list = sorted(list(set(string.ascii_uppercase) - set(pos_alphabet


positive_labels = len(LETTER_p2[LETTER_p2['letter'] == 1])/len(LETTER_p2['let
negative_labels = len(LETTER_p2[LETTER_p2['letter'] == 0])/len(LETTER_p2['let

print("% of negative labels:", negative_labels)
print("% of positive labels:", positive_labels)
```

```
% of negative labels: 50.3
% of positive labels: 49.7
```

Experiments

Random Forests

In [13]:
```
def split_data(data, column):

    Y = data[column]
    X = data.drop([column], axis=1)

    X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size=5000

    return X_train, X_test, y_train, y_test
```

In [14]:
```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score, accuracy_score, roc_auc_score
from sklearn.model_selection import train_test_split
```

In [36]:
```
ADULT_metric = []
COV_type_metric = []
LETTER_p1_metric = []
LETTER_p2_metric = []


accuracy_metric = []
f1_score_metric = []
roc_auc_score_metric = []


datalist = [COV_type_data, ADULT_one_hot_data, LETTER_p1, LETTER_p2]


for ind, data in enumerate(datalist):
    for i in range(5):

        print('Start of trial', i+1)

        # COV_type data


        if ind == 0:
            print('At COV_type_data')
            dataset = 'COV_type_data'
```

```
        X_train, X_test, y_train, y_test = split_data(data, 'forest_cover

    # ADULT_data

    elif ind == 1:
        print('At ADULT_data')
        dataset = 'ADULT_data'
        X_train, X_test, y_train, y_test = split_data(data, 'target_incom

    # LETTER.p1 data

    if ind == 2:
        print('At LETTER_p1')
        dataset = 'LETTER_p1'
        X_train, X_test, y_train, y_test = split_data(data, 'letter')

    # LETTER.p2 data
    if ind == 3:
        print('At LETTER_p2')
        dataset = 'LETTER_p2'
        X_train, X_test, y_train, y_test = split_data(data, 'letter')

    pipe = Pipeline([('classifier', RandomForestClassifier())
                     ])

    search_space = [
        { 'classifier': [RandomForestClassifier(n_estimators = 1024, clas
          'classifier__max_features': [1, 2, 4, 6, 8, 12, 16]
        }
    ]

# Create grid search
    clf = GridSearchCV(pipe, search_space, cv=StratifiedKFold(n_splits=5)
                scoring=['accuracy', 'roc_auc', 'f1'], refit='accuracy',
                verbose=0)

# Fit grid search
    best_model = clf.fit(X_train, y_train)

    # print(best_model.cv_results_['rank_test_accuracy'])

    best_acc_param = best_model.cv_results_['params'][ np.argmin(best_mod
    best_auc_param = best_model.cv_results_['params'][np.argmin(best_mode
    best_f1_param = best_model.cv_results_['params'][np.argmin(best_model

    # print(best_acc_param)

# Train 3 models using the 5000 samples and each of the 3 best parameter

# Tuned for accuracy
    acc_model = best_acc_param['classifier'].fit(X_train, y_train)
```

```
# Tuned for roc-auc score
    auc_model = best_auc_param['classifier'].fit(X_train, y_train)

# Tuned for f1 score
    f1_model =  best_f1_param['classifier'].fit(X_train, y_train)



    # fit a classifier using that best param on the training set,
    # predict the training set, and record the corresponding training set

   # On Training data

    y_pred_acc_tr = acc_model.predict(X_train)
    y_pred_auc_tr = auc_model.predict(X_train)
    y_pred_f1_tr = f1_model.predict(X_train)



    print('Trial ', i+1, ' raw training scores for', dataset)
    print(accuracy_score(y_train, y_pred_acc_tr))
    print(roc_auc_score(y_train, y_pred_auc_tr))
    print(f1_score(y_train, y_pred_f1_tr))


    # accuracy_metric_tr.append(accuracy_score(y_test, y_pred_acc_tr))
    # roc_auc_score_metric_tr.append(roc_auc_score(y_test, y_pred_auc_tr)
    # f1_score_metric_tr.append(f1_score(y_test, y_pred_f1_tr))


    # On Test data

    y_pred_acc = acc_model.predict(X_test)
    y_pred_auc = auc_model.predict(X_test)
    y_pred_f1 = f1_model.predict(X_test)


    print('Trial ', i+1, ' raw test scores for', dataset)
    print(accuracy_score(y_test, y_pred_acc))
    print(roc_auc_score(y_test, y_pred_auc))
    print(f1_score(y_test, y_pred_f1))


    accuracy_metric.append(accuracy_score(y_test, y_pred_acc))
    roc_auc_score_metric.append(roc_auc_score(y_test, y_pred_auc))
    f1_score_metric.append(f1_score(y_test, y_pred_f1))


    # Table 3
    if ind == 0:
        COV_type_metric.extend([accuracy_score(y_test, y_pred_acc), roc_a
                                f1_score(y_test, y_pred_f1)])
    elif ind == 1:
        ADULT_metric.extend([accuracy_score(y_test, y_pred_acc), roc_auc_
                                f1_score(y_test, y_pred_f1)])
    elif ind == 2:
        LETTER_p1_metric.extend([accuracy_score(y_test, y_pred_acc), roc_
                                f1_score(y_test, y_pred_f1)])

    elif ind == 3:
        LETTER_p2_metric.extend([accuracy_score(y_test, y_pred_acc), roc_
                                f1_score(y_test, y_pred_f1)])
```

```
        print("End of Trial", i+1)
        print('-----------------------------------------')
        print()
```

```
Start of trial 1
At COV_type_data
Trial  1  raw training scores for COV_type_data
1.0
1.0
1.0
Trial  1  raw test scores for COV_type_data
0.8235088852315576
0.8242443630436314
0.8250412610122587
End of Trial 1
-----------------------------------------

Start of trial 2
At COV_type_data
Trial  2  raw training scores for COV_type_data
1.0
1.0
1.0
Trial  2  raw test scores for COV_type_data
0.8185610716443408
0.8188204130922623
0.8167814072862484
End of Trial 2
-----------------------------------------

Start of trial 3
At COV_type_data
Trial  3  raw training scores for COV_type_data
1.0
1.0
1.0
Trial  3  raw test scores for COV_type_data
0.8249880210828942
0.8252495725808819
0.8232435076719146
End of Trial 3
-----------------------------------------

Start of trial 4
At COV_type_data
Trial  4  raw training scores for COV_type_data
1.0
1.0
1.0
Trial  4  raw test scores for COV_type_data
0.8184343381735103
0.8185831637591299
0.8158103295844003
End of Trial 4
-----------------------------------------

Start of trial 5
At COV_type_data
Trial  5  raw training scores for COV_type_data
1.0
1.0
1.0
Trial  5  raw test scores for COV_type_data
0.8223439789448831
```

```
0.8228222678883678
0.8220733878595672
End of Trial 5
----------------------------------------

Start of trial 1
At ADULT_data
Trial  1  raw training scores for ADULT_data
1.0
1.0
1.0
Trial  1  raw test scores for ADULT_data
0.8524001306193534
0.7660726804913083
0.6624626617988715
End of Trial 1
----------------------------------------

Start of trial 2
At ADULT_data
Trial  2  raw training scores for ADULT_data
1.0
1.0
1.0
Trial  2  raw test scores for ADULT_data
0.8542505714596713
0.7645542052930975
0.6632011402699757
End of Trial 2
----------------------------------------

Start of trial 3
At ADULT_data
Trial  3  raw training scores for ADULT_data
1.0
1.0
1.0
Trial  3  raw test scores for ADULT_data
0.8493886288596205
0.7713484440554236
0.6646740447532111
End of Trial 3
----------------------------------------

Start of trial 4
At ADULT_data
Trial  4  raw training scores for ADULT_data
1.0
1.0
1.0
Trial  4  raw test scores for ADULT_data
0.8493886288596205
0.771186018515124
0.6642943792964011
End of Trial 4
----------------------------------------

Start of trial 5
At ADULT_data
Trial  5  raw training scores for ADULT_data
1.0
1.0
1.0
Trial  5  raw test scores for ADULT_data
0.8520735822357679
0.7665547732446949
0.661856183130132
End of Trial 5
----------------------------------------
```

```
Start of trial 1
At LETTER_p1
Trial  1  raw training scores for LETTER_p1
1.0
1.0
1.0
Trial  1  raw test scores for LETTER_p1
0.9883333333333333
0.8560214407429669
0.8205128205128205
End of Trial 1
-----------------------------------------

Start of trial 2
At LETTER_p1
Trial  2  raw training scores for LETTER_p1
1.0
1.0
1.0
Trial  2  raw test scores for LETTER_p1
0.9876666666666667
0.8620295088239514
0.8198636806231743
End of Trial 2
-----------------------------------------

Start of trial 3
At LETTER_p1
Trial  3  raw training scores for LETTER_p1
1.0
1.0
1.0
Trial  3  raw test scores for LETTER_p1
0.9913333333333333
0.8983907666627527
0.87
End of Trial 3
-----------------------------------------

Start of trial 4
At LETTER_p1
Trial  4  raw training scores for LETTER_p1
1.0
1.0
1.0
Trial  4  raw test scores for LETTER_p1
0.9880666666666666
0.8636172005691807
0.8186423505572443
End of Trial 4
-----------------------------------------

Start of trial 5
At LETTER_p1
Trial  5  raw training scores for LETTER_p1
1.0
1.0
1.0
Trial  5  raw test scores for LETTER_p1
0.9894
0.8781960804110016
0.8408408408408408
End of Trial 5
-----------------------------------------

Start of trial 1
At LETTER_p2
Trial  1  raw training scores for LETTER_p2
```

```
1.0
1.0
1.0
Trial  1  raw test scores for LETTER_p2
0.9462
0.9462164727968981
0.9462179273575473
End of Trial 1
-----------------------------------------

Start of trial 2
At LETTER_p2
Trial  2  raw training scores for LETTER_p2
1.0
1.0
1.0
Trial  2  raw test scores for LETTER_p2
0.9444
0.9443367686451851
0.9435265438786566
End of Trial 2
-----------------------------------------

Start of trial 3
At LETTER_p2
Trial  3  raw training scores for LETTER_p2
1.0
1.0
1.0
Trial  3  raw test scores for LETTER_p2
0.9478
0.9478312344116951
0.9476918965862783
End of Trial 3
-----------------------------------------

Start of trial 4
At LETTER_p2
Trial  4  raw training scores for LETTER_p2
1.0
1.0
1.0
Trial  4  raw test scores for LETTER_p2
0.9465333333333333
0.9465191443063089
0.9461383478844863
End of Trial 4
-----------------------------------------

Start of trial 5
At LETTER_p2
Trial  5  raw training scores for LETTER_p2
1.0
1.0
1.0
Trial  5  raw test scores for LETTER_p2
0.9452
0.9453153254068586
0.9451048484039
End of Trial 5
-----------------------------------------
```

For Table 2

In [37]:
```
print('Accuracy metric values across all datasets, across 5 trials, for RF:')
print(accuracy_metric)
```

```
print()

print()
print('F-score metric values across all datasets, across 5 trials, for RF:')
print(f1_score_metric)

print()

print('ROC_AUC metric values across all datasets, across 5 trials, for RF:')
print(roc_auc_score_metric)

print()

print('Average scores for each metric: ')
print('ACC:', sum(accuracy_metric)/len(accuracy_metric))
print('FSC:', sum(f1_score_metric)/len(f1_score_metric))
print('ROC_AUC:', sum(roc_auc_score_metric)/len(roc_auc_score_metric))
```

```
Accuracy metric values across all datasets, across 5 trials, for RF:
[0.8235088852315576, 0.8185610716443408, 0.8249880210828942, 0.818434338173510
3, 0.8223439789448831, 0.8524001306193534, 0.8542505714596713, 0.8493886288596
205, 0.8493886288596205, 0.8520735822357679, 0.9883333333333333, 0.98766666666
66667, 0.9913333333333333, 0.9880666666666666, 0.9894, 0.9462, 0.9444, 0.9478,
0.9465333333333333, 0.9452]


F-score metric values across all datasets, across 5 trials, for RF:
[0.8250412610122587, 0.8167814072862484, 0.8232435076719146, 0.815810329584400
3, 0.8220733878595672, 0.6624626617988715, 0.6632011402699757, 0.6646740447532
111, 0.6642943792964011, 0.661856183130132, 0.8205128205128205, 0.819863680623
1743, 0.87, 0.8186423505572443, 0.8408408408408408, 0.9462179273575473, 0.9435
265438786566, 0.9476918965862783, 0.9461383478844863, 0.9451048484039]

ROC_AUC metric values across all datasets, across 5 trials, for RF:
[0.8242443630436314, 0.8188204130922623, 0.8252495725808819, 0.818583163759129
9, 0.8228222678883678, 0.7660726804913083, 0.7645542052930975, 0.7713484440554
236, 0.771186018515124, 0.7665547732446949, 0.8560214407429669, 0.862029508823
9514, 0.8983907666627527, 0.8636172005691807, 0.8781960804110016, 0.9462164727
968981, 0.9443367686451851, 0.9478312344116951, 0.9465191443063089, 0.94531532
54068586]

Average scores for each metric:
ACC: 0.9020135585222275
FSC: 0.8158988779653964
ROC_AUC: 0.8518954922370361
```

Write the 20 values for each metric to a .csv to do p-test comparisons.

In [44]:
```
with open('Table_2_p_test', 'a') as f:

    # using csv.writer method from CSV package
    write = csv.writer(f)

    write.writerow(accuracy_metric)

    write.writerow(f1_score_metric)

    write.writerow(roc_auc_score_metric)
```

For Table 3

In [46]:
```
print('COV_type')
print('Metric values across 5 trials, for RF:')
print(COV_type_metric)

print()
```

```
print('ADULT')
print('Metric values across 5 trials, for RF:')
print(ADULT_metric)

print()

print('LETTER.p1')
print('Metric values across 5 trials, for RF:')
print(LETTER_p1_metric)

print()

print('LETTER.p2')
print('Metric values across 5 trials, for RF:')
print(LETTER_p2_metric)
```

```
COV_type
Metric values across 5 trials, for RF:
[0.8235088852315576, 0.8242443630436314, 0.8250412610122587, 0.818561071644340
8, 0.8188204130922623, 0.8167814072862484, 0.8249880210828942, 0.8252495725808
819, 0.8232435076719146, 0.8184343381735103, 0.8185831637591299, 0.81581032958
44003, 0.8223439789448831, 0.8228222678883678, 0.8220733878595672]

ADULT
Metric values across 5 trials, for RF:
[0.8524001306193534, 0.7660726804913083, 0.6624626617988715, 0.854250571459671
3, 0.7645542052930975, 0.6632011402699757, 0.8493886288596205, 0.7713484440554
236, 0.6646740447532111, 0.8493886288596205, 0.771186018515124, 0.664294379296
4011, 0.8520735822357679, 0.7665547732446949, 0.661856183130132]

LETTER.p1
Metric values across 5 trials, for RF:
[0.9883333333333333, 0.8560214407429669, 0.8205128205128205, 0.987666666666666
7, 0.8620295088239514, 0.8198636806231743, 0.9913333333333333, 0.8983907666627
527, 0.87, 0.9880666666666666, 0.8636172005691807, 0.8186423505572443, 0.9894,
0.8781960804110016, 0.8408408408408408]

LETTER.p2
Metric values across 5 trials, for RF:
[0.9462, 0.9462164727968981, 0.9462179273575473, 0.9444, 0.9443367686451851,
0.9435265438786566, 0.9478, 0.9478312344116951, 0.9476918965862783, 0.94653333
33333333, 0.9465191443063089, 0.9461383478844863, 0.9452, 0.9453153254068586,
0.9451048484039]
```

Write the 15 values for each metric to a .csv to do p-test comparisons.

In [47]:
```
with open('Table_3_p_test', 'a') as f:

    # using csv.writer method from CSV package
    write = csv.writer(f)

    write.writerow(COV_type_metric)

    write.writerow(ADULT_metric)

    write.writerow(LETTER_p1_metric)

    write.writerow(LETTER_p2_metric)
```

In [39]:
```
print('Average metric scores for each dataset across 5 trials: ')

print()
print('COV_type:', sum(COV_type_metric)/len(COV_type_metric))
print()

print('ADULT:', sum(ADULT_metric)/len(ADULT_metric))
```

```
print()

print('LETTER.p1:', sum(LETTER_p1_metric)/len(LETTER_p1_metric))
print()

print('LETTER.p2:', sum(LETTER_p2_metric)/len(LETTER_p2_metric))
```

```
Average metric scores for each dataset across 5 trials:

COV_type: 0.8213670645903897

ADULT: 0.7609137381921515

LETTER.p1: 0.8981943126495955

LETTER.p2: 0.9459354562007432
```

In [ ]:

In [53]:
```
import pandas as pd
import numpy as np
import csv
import string
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
```

This notebook runs k-NN over 4 problem sets across 5 trials. Table 2 and Table 3 values are recorded at each iteration of the for loop.

Datasets

ADULT

In [2]:
```
ADULT_data = pd.read_csv('adult.data.csv', names = ['age', 'workclass', 'fnlw
                      'relationship', 'race', 'sex', 'capital-gain', 'capital
                      'native-country', 'target_income'
                     ])

ADULT_data['target_income'] = ADULT_data['target_income'].str.strip()

ADULT_data['target_income'] = ADULT_data.target_income.map( {'<=50K':0 , '>50

ADULT_one_hot_data = pd.get_dummies(ADULT_data,
                                    columns = ['workclass', 'education', 'mar
                                               'occupation', 'relationship', 'ra
                                    prefix = ['workclass', 'education', 'mari
                                               'occupation', 'relationship', 'ra

ADULT_one_hot_data = ADULT_one_hot_data.drop(['workclass_ ?',
                                              'occupation_ ?', 'native-countr


ADULT_one_hot_data[['age', 'fnlwgt', 'education-num', 'capital-gain',
    'capital-loss', 'hours-per-week']] = StandardScaler().fit_transform(ADULT_



ADULT_one_hot_data
```

Out[2]:

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	target_income	wor F
0	0.030671	-1.063611	1.134739	0.148453	-0.21666	-0.035429	0	
1	0.837109	-1.008707	1.134739	-0.145920	-0.21666	-2.222153	0	
2	-0.042642	0.245079	-0.420060	-0.145920	-0.21666	-0.035429	0	
3	1.057047	0.425801	-1.197459	-0.145920	-0.21666	-0.035429	0	
4	-0.775768	1.408176	1.134739	-0.145920	-0.21666	-0.035429	0	
...	...	...	...	...	...	...	...	
32556	-0.849080	0.639741	0.746039	-0.145920	-0.21666	-0.197409	0	
32557	0.103983	-0.335433	-0.420060	-0.145920	-0.21666	-0.035429	1	
32558	1.423610	-0.358777	-0.420060	-0.145920	-0.21666	-0.035429	0	
32559	-1.215643	0.110960	-0.420060	-0.145920	-0.21666	-1.655225	0	

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	target_income	wor F
32560	0.983734	0.929893	-0.420060	1.888424	-0.21666	-0.035429	1	

32561 rows × 106 columns

Balance of dataset

In [3]:
```
positive_labels = ADULT_data['target_income'].value_counts()[1]/ ADULT_data['
negative_labels = ADULT_data['target_income'].value_counts()[0]/ ADULT_data['
print("% of negative labels:", negative_labels)
print("% of positive labels:", positive_labels)
```

```
% of negative labels: 75.91904425539757
% of positive labels: 24.080955744602438
```

Unbalanced dataset

COV_type data

In [4]:
```
COV_type_data = pd.read_csv('covtype.data.gz', header = None)

cols = [c for c in COV_type_data.columns]
cols[-1] = 'forest_cover'
COV_type_data.columns = cols



largest_class = COV_type_data['forest_cover'].value_counts().idxmax()

COV_type_data.loc[COV_type_data['forest_cover'] != largest_class, 'forest_cov
COV_type_data.loc[COV_type_data['forest_cover'] == largest_class, 'forest_cov



COV_type_data.iloc[:, :-1] = StandardScaler().fit_transform(COV_type_data.ilo

COV_type_data
```

Out[4]:

	0	1	2	3	4	5	6	
0	-1.297805	-0.935157	-1.482820	-0.053767	-0.796273	-1.180146	0.330743	0.4391
1	-1.319235	-0.890480	-1.616363	-0.270188	-0.899197	-1.257106	0.293388	0.5908
2	-0.554907	-0.148836	-0.681563	-0.006719	0.318742	0.532212	0.816364	0.7426
3	-0.622768	-0.005869	0.520322	-0.129044	1.227908	0.474492	0.965786	0.7426
4	-1.301377	-0.988770	-1.616363	-0.547771	-0.813427	-1.256464	0.293388	0.5403
...	...	...	...	...	...	...	...	
581007	-2.012130	-0.023740	0.787408	-0.867697	-0.504653	-1.437962	1.040496	0.6920
581008	-2.029988	-0.032675	0.653865	-0.952383	-0.590424	-1.446299	1.040496	0.6920
581009	-2.047847	0.029873	0.386780	-0.985317	-0.676194	-1.449506	0.891075	0.8944
581010	-2.054990	0.128163	0.119694	-0.985317	-0.710502	-1.449506	0.666942	1.0967
581011	-2.058562	0.083486	-0.147392	-0.985317	-0.727656	-1.464256	0.704298	1.0461

581012 rows × 55 columns

Balance of dataset

Treat largest class as positive class. The rest are negative.

In [5]:
```
# positive_labels = len(COV_type_data[COV_type_data['Forest cover'] == 7])/le

positive_labels = COV_type_data['forest_cover'].value_counts().max()/len(COV_
negative_labels = len(COV_type_data[COV_type_data['forest_cover'] != COV_type_

print("% of negative labels:", negative_labels)
print("% of positive labels:", positive_labels)
```

```
% of negative labels: 48.75992234239568
% of positive labels: 51.240077657604324
```

LETTER

In [8]:
```
LETTER_p1 = pd.read_csv('letter-recognition.data', header = None)

cols = [c for c in LETTER_p1.columns]
cols[0] = 'letter'
LETTER_p1.columns = cols


LETTER_p1.loc[:, LETTER_p1.columns != 'letter'] = StandardScaler().fit_transf
LETTER_p1
```

Out[8]:

	letter	1	2	3	4	5	6	7
0	T	-1.057698	0.291877	-1.053277	-0.164704	-1.144013	0.544130	2.365097
1	I	0.510385	1.502358	-1.053277	0.719730	-0.687476	1.531305	-1.075326
2	D	-0.012309	1.199738	0.435910	1.161947	1.138672	1.531305	-0.645273
3	N	1.555774	1.199738	0.435910	0.277513	-0.230939	-0.936631	0.644886
4	G	-1.057698	-1.826464	-1.053277	-1.933571	-1.144013	0.544130	-0.645273
...	...	...	...	...	...	...	...	...
19995	D	-1.057698	-1.523844	-1.053277	-1.049137	-0.687476	0.050543	-0.215220
19996	C	1.555774	0.897117	1.428701	1.161947	0.225598	-1.430218	0.214833
19997	T	1.033079	0.594497	0.435910	0.719730	0.682135	-0.443044	1.504991
19998	S	-1.057698	-1.221224	-0.556881	-1.491354	-1.144013	0.544130	-0.215220
19999	A	-0.012309	0.594497	0.435910	0.277513	-0.687476	1.037718	-1.075326

20000 rows × 17 columns

Letter.p1 - treat O as positive class, rest as negative

Unbalanced dataset

In [9]:
```
O_list = ['O']


LETTER_p1.loc[~LETTER_p1['letter'].isin(O_list), 'letter'] = 0
LETTER_p1.loc[LETTER_p1['letter'].isin(O_list), 'letter'] = 1
```

```
LETTER_p1['letter'].value_counts()
```

Out[9]:
```
0    19247
1      753
Name: letter, dtype: int64
```

In [10]:
```
positive_labels = len(LETTER_p1[LETTER_p1['letter'] == 1])/len(LETTER_p1['let
negative_labels = len(LETTER_p1[LETTER_p1['letter'] == 0])/len(LETTER_p1['let

print("% of negative labels:", negative_labels)
print("% of positive labels:", positive_labels)
```

```
% of negative labels: 96.235
% of positive labels: 3.765
```

Letter.p2 - treat A-M as positive class, rest as negative

In [13]:
```
LETTER_p2 = pd.read_csv('letter-recognition.data', header = None)

cols = [c for c in LETTER_p2.columns]
cols[0] = 'letter'
LETTER_p2.columns = cols


LETTER_p2.loc[:, LETTER_p2.columns != 'letter'] = StandardScaler().fit_transf

pos_alphabet_list = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', '
neg_alphabet_list = sorted(list(set(string.ascii_uppercase) - set(pos_alphabe




LETTER_p2.loc[LETTER_p2['letter'].isin(pos_alphabet_list), 'letter'] = 1
LETTER_p2.loc[LETTER_p2['letter'].isin(neg_alphabet_list), 'letter'] = 0

LETTER_p1["letter"] = LETTER_p1["letter"].astype(str).astype(int)
LETTER_p2["letter"] = LETTER_p2["letter"].astype(str).astype(int)

LETTER_p2
```

Out[13]:

	letter	1	2	3	4	5	6	7
0	0	-1.057698	0.291877	-1.053277	-0.164704	-1.144013	0.544130	2.365097
1	1	0.510385	1.502358	-1.053277	0.719730	-0.687476	1.531305	-1.075326
2	1	-0.012309	1.199738	0.435910	1.161947	1.138672	1.531305	-0.645273
3	0	1.555774	1.199738	0.435910	0.277513	-0.230939	-0.936631	0.644886
4	1	-1.057698	-1.826464	-1.053277	-1.933571	-1.144013	0.544130	-0.645273
...	...	...	...	...	...	...	...	...
19995	1	-1.057698	-1.523844	-1.053277	-1.049137	-0.687476	0.050543	-0.215220
19996	1	1.555774	0.897117	1.428701	1.161947	0.225598	-1.430218	0.214833
19997	0	1.033079	0.594497	0.435910	0.719730	0.682135	-0.443044	1.504991
19998	0	-1.057698	-1.221224	-0.556881	-1.491354	-1.144013	0.544130	-0.215220
19999	1	-0.012309	0.594497	0.435910	0.277513	-0.687476	1.037718	-1.075326

20000 rows × 17 columns

Well-balanced dataset

In [15]:
```
pos_alphabet_list = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L
neg_alphabet_list = sorted(list(set(string.ascii_uppercase) - set(pos_alphabet


positive_labels = len(LETTER_p2[LETTER_p2['letter'] == 1])/len(LETTER_p2['let
negative_labels = len(LETTER_p2[LETTER_p2['letter'] == 0])/len(LETTER_p2['let

print("% of negative labels:", negative_labels)
print("% of positive labels:", positive_labels)
```

```
% of negative labels: 50.3
% of positive labels: 49.7
```

Experiment - KNN over 4 datasets over 5 trials

In [16]:
```
def split_data(data, column):

    Y = data[column]
    X = data.drop([column], axis=1)

    X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size=5000

    return X_train, X_test, y_train, y_test
```

In [17]:
```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score, accuracy_score, roc_auc_score
from sklearn.model_selection import train_test_split
```

In [48]:
```
accuracy_metric = []
f1_score_metric = []
roc_auc_score_metric = []

ADULT_metric = []
COV_type_metric = []
LETTER_p1_metric = []
LETTER_p2_metric = []


datalist = [COV_type_data, ADULT_one_hot_data, LETTER_p1, LETTER_p2]


for ind, data in enumerate(datalist):
    for i in range(5):

        print('Start of trial', i+1)

        # COV_type data


        if ind == 0:
            print('At COV_type_data')
            dataset = 'COV_type_data'
            X_train, X_test, y_train, y_test = split_data(data, 'forest_cover


        # ADULT_data
```

```
elif ind == 1:
    print('At ADULT_data')
    dataset = 'ADULT_data'
    X_train, X_test, y_train, y_test = split_data(data, 'target_incom

# LETTER.p1 data

if ind == 2:
    print('At LETTER_p1')
    dataset = 'LETTER_p1'
    X_train, X_test, y_train, y_test = split_data(data, 'letter')


# LETTER.p2 data
if ind == 3:
    print('At LETTER_p2')
    dataset = 'LETTER_p2'
    X_train, X_test, y_train, y_test = split_data(data, 'letter')



pipe = Pipeline([('classifier', KNeighborsClassifier())
     ])



search_space = [
    { 'classifier': [KNeighborsClassifier(p = 2)],
      'classifier__n_neighbors': list(range(1,102,4)),
      'classifier__metric': ['euclidean'],
      'classifier__weights': ['uniform', 'distance']
    }]


# Create grid search
clf = GridSearchCV(pipe, search_space, cv=StratifiedKFold(n_splits=5)
           scoring=['accuracy', 'roc_auc', 'f1'], refit='accuracy',
           verbose=0, n_jobs = -1)


# Fit grid search
best_model = clf.fit(X_train, y_train)

# Get best hyperparameters for accuracy, roc_auc score, f1_score

best_acc_param = best_model.cv_results_['params'][ np.argmin(best_mod
best_auc_param = best_model.cv_results_['params'][np.argmin(best_mode
best_f1_param = best_model.cv_results_['params'][np.argmin(best_model



# Train 3 models using the 5000 samples and each of the 3 best parame

# Tuned for accuracy
acc_model = best_acc_param['classifier'].fit(X_train, y_train)

# Tuned for roc-auc score
auc_model = best_auc_param['classifier'].fit(X_train, y_train)

# Tuned for f1 score
f1_model =  best_f1_param['classifier'].fit(X_train, y_train)
```

```
        # fit a classifier using that best param on the training set,
        # predict the training set, and record the corresponding training set

       # On Training data

        y_pred_acc_tr = acc_model.predict(X_train)
        y_pred_auc_tr = auc_model.predict(X_train)
        y_pred_f1_tr = f1_model.predict(X_train)



        print('Trial ', i+1, ' raw training scores for', dataset)

         # Raw train accuracy score
        print(accuracy_score(y_train, y_pred_acc_tr))

        # Raw train roc_auc score
        print(roc_auc_score(y_train, y_pred_auc_tr))

         # Raw train f1_score
        print(f1_score(y_train, y_pred_f1_tr))



        # On Test data

        y_pred_acc = acc_model.predict(X_test)

        y_pred_auc = auc_model.predict(X_test)

        y_pred_f1 = f1_model.predict(X_test)


        print('Trial ', i+1, ' raw test scores for', dataset)

         # Raw test accuracy score
        print(accuracy_score(y_test, y_pred_acc))

        # Raw test roc_auc score
        print(roc_auc_score(y_test, y_pred_auc))

        # Raw test f1_score
        print(f1_score(y_test, y_pred_f1))

         # Append raw test scores to list to generate Table 2 values
        accuracy_metric.append(accuracy_score(y_test, y_pred_acc))
        roc_auc_score_metric.append(roc_auc_score(y_test, y_pred_auc))
        f1_score_metric.append(f1_score(y_test, y_pred_f1))


          # For Table 3
        if ind == 0:
            COV_type_metric.extend([accuracy_score(y_test, y_pred_acc), roc_a
                                    f1_score(y_test, y_pred_f1)])
        elif ind == 1:
            ADULT_metric.extend([accuracy_score(y_test, y_pred_acc), roc_auc_
                                    f1_score(y_test, y_pred_f1)])
        elif ind == 2:
            LETTER_p1_metric.extend([accuracy_score(y_test, y_pred_acc), roc_
                                    f1_score(y_test, y_pred_f1)])

        elif ind == 3:
            LETTER_p2_metric.extend([accuracy_score(y_test, y_pred_acc), roc_
                                    f1_score(y_test, y_pred_f1)])
```

```
        print("End of Trial", i+1)
        print('-----------------------------------------')
        print()
```

```
Start of trial 1
At COV_type_data
Trial  1  raw training scores for COV_type_data
1.0
1.0
1.0
Trial  1  raw test scores for COV_type_data
0.7773657493246668
0.7776871484457607
0.7759436434666537
End of Trial 1
-----------------------------------------

Start of trial 2
At COV_type_data
Trial  2  raw training scores for COV_type_data
1.0
1.0
1.0
Trial  2  raw test scores for COV_type_data
0.7752616264938925
0.7752553645210009
0.7708221947618323
End of Trial 2
-----------------------------------------

Start of trial 3
At COV_type_data
Trial  3  raw training scores for COV_type_data
1.0
1.0
1.0
Trial  3  raw test scores for COV_type_data
0.7866433338194343
0.7869815425421214
0.7853779845481091
End of Trial 3
-----------------------------------------

Start of trial 4
At COV_type_data
Trial  4  raw training scores for COV_type_data
1.0
1.0
1.0
Trial  4  raw test scores for COV_type_data
0.7746661527884836
0.7754884432741864
0.7776831914824376
End of Trial 4
-----------------------------------------

Start of trial 5
At COV_type_data
Trial  5  raw training scores for COV_type_data
1.0
1.0
1.0
Trial  5  raw test scores for COV_type_data
0.7766747914973994
0.7773724385257351
0.7785958933581179
End of Trial 5
```

```
----------------------------------------

Start of trial 1
At ADULT_data
Trial  1  raw training scores for ADULT_data
1.0
1.0
1.0
Trial  1  raw test scores for ADULT_data
0.8377780196654693
0.7423299210968046
0.6227322588811071
End of Trial 1
----------------------------------------

Start of trial 2
At ADULT_data
Trial  2  raw training scores for ADULT_data
1.0
1.0
1.0
Trial  2  raw test scores for ADULT_data
0.8377054533580058
0.7439800499855918
0.6255963840294635
End of Trial 2
----------------------------------------

Start of trial 3
At ADULT_data
Trial  3  raw training scores for ADULT_data
0.838
0.7463818691923787
0.632486388384755
Trial  3  raw test scores for ADULT_data
0.8369797902833714
0.7435189713183991
0.6239223235958817
End of Trial 3
----------------------------------------

Start of trial 4
At ADULT_data
Trial  4  raw training scores for ADULT_data
1.0
1.0
1.0
Trial  4  raw test scores for ADULT_data
0.8361452777475418
0.7421940897545529
0.623603933988998
End of Trial 4
----------------------------------------

Start of trial 5
At ADULT_data
Trial  5  raw training scores for ADULT_data
1.0
1.0
1.0
Trial  5  raw test scores for ADULT_data
0.8348028010594681
0.7370038986558887
0.615813011560206
End of Trial 5
----------------------------------------

Start of trial 1
At LETTER_p1
```

```
Trial  1  raw training scores for LETTER_p1
1.0
1.0
1.0
Trial  1  raw test scores for LETTER_p1
0.9893333333333333
0.9359170166730304
0.8646362098138749
End of Trial 1
----------------------------------------

Start of trial 2
At LETTER_p1
Trial  2  raw training scores for LETTER_p1
1.0
1.0
1.0
Trial  2  raw test scores for LETTER_p1
0.9900666666666667
0.9549933110680774
0.8723221936589546
End of Trial 2
----------------------------------------

Start of trial 3
At LETTER_p1
Trial  3  raw training scores for LETTER_p1
1.0
1.0
1.0
Trial  3  raw test scores for LETTER_p1
0.9898666666666667
0.9507199928598129
0.8735440931780367
End of Trial 3
----------------------------------------

Start of trial 4
At LETTER_p1
Trial  4  raw training scores for LETTER_p1
1.0
1.0
1.0
Trial  4  raw test scores for LETTER_p1
0.9913333333333333
0.9323411783247382
0.8826714801444042
End of Trial 4
----------------------------------------

Start of trial 5
At LETTER_p1
Trial  5  raw training scores for LETTER_p1
1.0
1.0
1.0
Trial  5  raw test scores for LETTER_p1
0.9888666666666667
0.9237303792098314
0.8480436760691538
End of Trial 5
----------------------------------------

Start of trial 1
At LETTER_p2
Trial  1  raw training scores for LETTER_p2
1.0
1.0
1.0
```

```
Trial  1  raw test scores for LETTER_p2
0.9556
0.9556005262294175
0.9556414013587319
End of Trial 1
-----------------------------------------

Start of trial 2
At LETTER_p2
Trial  2  raw training scores for LETTER_p2
1.0
1.0
1.0
Trial  2  raw test scores for LETTER_p2
0.9488666666666666
0.9488722922252363
0.9487813021702838
End of Trial 2
-----------------------------------------

Start of trial 3
At LETTER_p2
Trial  3  raw training scores for LETTER_p2
1.0
1.0
1.0
Trial  3  raw test scores for LETTER_p2
0.9537333333333333
0.9537293026414745
0.9532974427994616
End of Trial 3
-----------------------------------------

Start of trial 4
At LETTER_p2
Trial  4  raw training scores for LETTER_p2
1.0
1.0
1.0
Trial  4  raw test scores for LETTER_p2
0.9532666666666667
0.9532509464739012
0.9527213866594726
End of Trial 4
-----------------------------------------

Start of trial 5
At LETTER_p2
Trial  5  raw training scores for LETTER_p2
1.0
1.0
1.0
Trial  5  raw test scores for LETTER_p2
0.9518
0.9518224963519245
0.9517259798357482
End of Trial 5
-----------------------------------------
```

For Table 2

In [70]:

```
print('Accuracy metric values across all datasets, across 5 trials, for KNN:'
print(accuracy_metric)

print()

print()
```

```
print('F-score metric values across all datasets, across 5 trials, for KNN:')
print(f1_score_metric)

print()

print('ROC_AUC metric values across all datasets, across 5 trials, for KNN:')
print(roc_auc_score_metric)

print()

print('Average scores for each metric: ')
print('ACC:', sum(accuracy_metric)/len(accuracy_metric))
print('FSC:', sum(f1_score_metric)/len(f1_score_metric))
print('ROC_AUC:', sum(roc_auc_score_metric)/len(roc_auc_score_metric))
```

```
Accuracy metric values across all datasets, across 5 trials, for KNN:
[0.7773657493246668, 0.7752616264938925, 0.7866433338194343, 0.774666152788483
6, 0.7766747914973994, 0.8377780196654693, 0.8377054533580058, 0.8369797902833
714, 0.8361452777475418, 0.8348028010594681, 0.9893333333333333, 0.99006666666
66667, 0.9898666666666667, 0.9913333333333333, 0.9888666666666667, 0.9556, 0.9
488666666666666, 0.9537333333333333, 0.9532666666666667, 0.9518]


F-score metric values across all datasets, across 5 trials, for KNN:
[0.7759436434666537, 0.7708221947618323, 0.7853779845481091, 0.777683191482437
6, 0.7785958933581179, 0.6227322588811071, 0.6255963840294635, 0.6239223235958
817, 0.623603933988998, 0.615813011560206, 0.8646362098138749, 0.8723221936589
546, 0.8735440931780367, 0.8826714801444042, 0.8480436760691538, 0.9556414013587319, 0.9487813021702838, 0.9532974427994616, 0.9527213866594726, 0.951725979
8357482]

ROC_AUC metric values across all datasets, across 5 trials, for KNN:
[0.7776871484457607, 0.7752553645210009, 0.7869815425421214, 0.775488443274186
4, 0.7773724385257351, 0.7423299210968046, 0.7439800499855918, 0.7435189713183
991, 0.7421940897545529, 0.7370038986558887, 0.9359170166730304, 0.95499331106
80774, 0.9507199928598129, 0.9323411783247382, 0.9237303792098314, 0.955600526
2294175, 0.9488722922252363, 0.9537293026414745, 0.9532509464739012, 0.9518224
963519245]

Average scores for each metric:
ACC: 0.8893378164685533
FSC: 0.8051737992680466
ROC_AUC: 0.8531394655088743
```

In [72]:
```
with open('Table_2_p_test', 'a') as f:

    # using csv.writer method from CSV package
    write = csv.writer(f)

    write.writerow(accuracy_metric)

    write.writerow(f1_score_metric)

    write.writerow(roc_auc_score_metric)
```

For Table 3

In [74]:
```
print('COV_type')
print('Metric values across 5 trials, for KNN:')
print(COV_type_metric)

print()

print('ADULT')
print('Metric values across 5 trials, for KNN:')
```

```
print(ADULT_metric)

print()

print('LETTER.p1')
print('Metric values across 5 trials, for KNN:')
print(LETTER_p1_metric)

print()

print('LETTER.p2')
print('Metric values across 5 trials, for KNN:')
print(LETTER_p2_metric)
```

```
COV_type
Metric values across 5 trials, for KNN:
[0.7773657493246668, 0.7776871484457607, 0.7759436434666537, 0.775261626493892
5, 0.7752553645210009, 0.7708221947618323, 0.7866433338194343, 0.7869815425421
214, 0.7853779845481091, 0.7746661527884836, 0.7754884432741864, 0.77768319148
24376, 0.7766747914973994, 0.7773724385257351, 0.7785958933581179]

ADULT
Metric values across 5 trials, for KNN:
[0.8377780196654693, 0.7423299210968046, 0.6227322588811071, 0.837705453358005
8, 0.7439800499855918, 0.6255963840294635, 0.8369797902833714, 0.7435189713183
991, 0.6239223235958817, 0.8361452777475418, 0.7421940897545529, 0.62360393398
8998, 0.8348028010594681, 0.7370038986558887, 0.615813011560206]

LETTER.p1
Metric values across 5 trials, for KNN:
[0.9893333333333333, 0.9359170166730304, 0.8646362098138749, 0.990066666666666
7, 0.9549933110680774, 0.8723221936589546, 0.9898666666666667, 0.9507199928598
129, 0.8735440931780367, 0.9913333333333333, 0.9323411783247382, 0.88267148014
44042, 0.9888666666666667, 0.9237303792098314, 0.8480436760691538]

LETTER.p2
Metric values across 5 trials, for KNN:
[0.9556, 0.9556005262294175, 0.9556414013587319, 0.9488666666666666, 0.9488722
922252363, 0.9487813021702838, 0.9537333333333333, 0.9537293026414745, 0.95329
74427994616, 0.9532666666666667, 0.9532509464739012, 0.9527213866594726, 0.951
8, 0.9518224963519245, 0.9517259798357482]
```

Write the 15 values for each metric to a .csv to do p-test comparisons.

In [75]:
```
with open('Table_3_p_test', 'a') as f:

    # using csv.writer method from CSV package
    write = csv.writer(f)

    write.writerow(COV_type_metric)

    write.writerow(ADULT_metric)

    write.writerow(LETTER_p1_metric)

    write.writerow(LETTER_p2_metric)
```

In [76]:
```
print('Average metric scores for each dataset across 5 trials: ')

print()
print('COV_type:', sum(COV_type_metric)/len(COV_type_metric))
print()

print('ADULT:', sum(ADULT_metric)/len(ADULT_metric))
print()

print('LETTER.p1:', sum(LETTER_p1_metric)/len(LETTER_p1_metric))
```

```
print()

print('LETTER.p2:', sum(LETTER_p2_metric)/len(LETTER_p2_metric))
```

```
Average metric scores for each dataset across 5 trials:

COV_type: 0.778121299923322

ADULT: 0.7336070789987166

LETTER.p1: 0.9325590798444385

LETTER.p2: 0.9525806495608212
```

In [ ]:

In [23]:
```
import pandas as pd
from scipy import stats
```

For Table 2 p-test comparison

In [56]:
```
df = pd.read_csv('Table_2_p_test', delimiter=',', header = None)
df.insert(0, 'col', ['LR_acc', 'LR_f1', 'LR_roc', 'RF_acc', 'RF_f1', 'RF_roc'
df = df.set_index('col')
print(df)
```

```
                0         1         2         3         4         5         6
\
col
LR_acc   0.752082  0.749181  0.756198  0.747785  0.749609  0.810167  0.808643
LR_f1    0.753565  0.751327  0.758021  0.751306  0.754051  0.680039  0.680015
LR_roc   0.752708  0.749852  0.754912  0.748572  0.750519  0.819488  0.821004
RF_acc   0.823509  0.818561  0.824988  0.818434  0.822344  0.852400  0.854251
RF_f1    0.825041  0.816781  0.823244  0.815810  0.822073  0.662463  0.663201
RF_roc   0.824244  0.818820  0.825250  0.818583  0.822822  0.766073  0.764554
KNN_acc  0.777366  0.775262  0.786643  0.774666  0.776675  0.837778  0.837705
KNN_f1   0.775944  0.770822  0.785378  0.777683  0.778596  0.622732  0.625596
KNN_roc  0.777687  0.775255  0.786982  0.775488  0.777372  0.742330  0.743980

                7         8         9         10        11        12        13
\
col
LR_acc   0.809405  0.810239  0.810783  0.790533  0.742333  0.779800  0.719067
LR_f1    0.677829  0.681020  0.681566  0.244348  0.234162  0.233820  0.192101
LR_roc   0.818025  0.819795  0.821328  0.831482  0.780580  0.839912  0.794362
RF_acc   0.849389  0.849389  0.852074  0.988333  0.987667  0.991333  0.988067
RF_f1    0.664674  0.664294  0.661856  0.820513  0.819864  0.870000  0.818642
RF_roc   0.771348  0.771186  0.766555  0.856021  0.862030  0.898391  0.863617
KNN_acc  0.836980  0.836145  0.834803  0.989333  0.990067  0.989867  0.991333
KNN_f1   0.623922  0.623604  0.615813  0.864636  0.872322  0.873544  0.882671
KNN_roc  0.743519  0.742194  0.737004  0.935917  0.954993  0.950720  0.932341

                14        15        16        17        18        19
col
LR_acc   0.795000  0.727400  0.729000  0.726800  0.721867  0.730067
LR_f1    0.239050  0.731499  0.731524  0.731490  0.720899  0.733496
LR_roc   0.833993  0.727417  0.729119  0.727241  0.721912  0.730266
RF_acc   0.989400  0.946200  0.944400  0.947800  0.946533  0.945200
RF_f1    0.840841  0.946218  0.943527  0.947692  0.946138  0.945105
RF_roc   0.878196  0.946216  0.944337  0.947831  0.946519  0.945315
KNN_acc  0.988867  0.955600  0.948867  0.953733  0.953267  0.951800
KNN_f1   0.848044  0.955641  0.948781  0.953297  0.952721  0.951726
KNN_roc  0.923730  0.955601  0.948872  0.953729  0.953251  0.951822
```

I first conducted t-tests using t_test rel.

In [57]:
```
# RF - highest ACC
print('For ACC:')
print('RF vs LR acc')
print(stats.ttest_rel(df.loc['RF_acc'], df.loc['LR_acc']))


print('RF vs KNN acc')
print(stats.ttest_rel(df.loc['RF_acc'], df.loc['KNN_acc']))

print()

# RF - highest FSC
print('For FSC:')
print('RF vs LR FSC')
print(stats.ttest_rel(df.loc['RF_f1'], df.loc['LR_f1']))
```

```
print('RF vs KNN FSC')
print(stats.ttest_rel(df.loc['RF_f1'], df.loc['KNN_f1']))

print()


# KNN - highest ROC
print('For ROC:')

print('KNN vs LR ROC')
print(stats.ttest_rel(df.loc['KNN_roc'], df.loc['LR_roc']))

print('KNN vs RF ROC')
print(stats.ttest_rel(df.loc['KNN_roc'], df.loc['RF_roc']))
```

```
For ACC:
RF vs LR acc
Ttest_relResult(statistic=7.153980868866512, pvalue=8.456295042265277e-07)
RF vs KNN acc
Ttest_relResult(statistic=2.8288677195201037, pvalue=0.010726528244730313)

For FSC:
RF vs LR FSC
Ttest_relResult(statistic=3.972875940229491, pvalue=0.0008151666170288787)
RF vs KNN FSC
Ttest_relResult(statistic=1.3349652702105461, pvalue=0.19766416057965472)

For ROC:
KNN vs LR ROC
Ttest_relResult(statistic=2.858522244149977, pvalue=0.010052969377641703)
KNN vs RF ROC
Ttest_relResult(statistic=0.12491041744019583, pvalue=0.901906608991005)
```

I then conducted the same t-test analysis using t_test_ind

In [64]:

```
print('Using t_test_ind')
# RF - highest ACC
print('For ACC:')
print('RF vs LR acc')
print(stats.ttest_ind(df.loc['RF_acc'], df.loc['LR_acc']))


print('RF vs KNN acc')
print(stats.ttest_ind(df.loc['RF_acc'], df.loc['KNN_acc']))

print()

# RF - highest FSC
print('For FSC:')
print('RF vs LR FSC')
print(stats.ttest_ind(df.loc['RF_f1'], df.loc['LR_f1']))

print('RF vs KNN FSC')
print(stats.ttest_ind(df.loc['RF_f1'], df.loc['KNN_f1']))

print()


# KNN - highest ROC
print('For ROC:')

print('KNN vs LR ROC')
print(stats.ttest_ind(df.loc['KNN_roc'], df.loc['LR_roc']))

print('KNN vs RF ROC')
print(stats.ttest_ind(df.loc['KNN_roc'], df.loc['RF_roc']))
```

```
Using t_test_ind
For ACC:
RF vs LR acc
Ttest_indResult(statistic=7.9620388188564215, pvalue=1.2780017901507084e-09)
RF vs KNN acc
Ttest_indResult(statistic=0.5053226845432764, pvalue=0.6162507401123773)

For FSC:
RF vs LR FSC
Ttest_indResult(statistic=3.994816253121585, pvalue=0.00028676664996692364)
RF vs KNN FSC
Ttest_indResult(statistic=0.2943126401835593, pvalue=0.7701207869528658)

For ROC:
KNN vs LR ROC
Ttest_indResult(statistic=3.14819842574865, pvalue=0.0031924364536140546)
KNN vs RF ROC
Ttest_indResult(statistic=0.047133130059937973, pvalue=0.9626539676774699)
```

For Table 3 p-test Comparison

In [60]:

```
df_3 = pd.read_csv('Table_3_p_test', delimiter=',', header = None)
df_3.insert(0, 'col', ['LR_COV', 'LR_ADULT', 'LR_LETTER_p1', 'LR_LETTER_p2',
                       'RF_COV', 'RF_ADULT', 'RF_LETTER_p1', 'RF_LETTER_p2',
                       'KNN_COV', 'KNN_ADULT', 'KNN_LETTER_p1', 'KNN_LETTER_p2'
                      ])


df_3 = df_3.set_index('col')
print(df_3)
```

```
                    0         1         2         3         4         5   \
col
LR_COV          0.752082  0.752708  0.753565  0.749181  0.749852  0.751327
LR_ADULT        0.810167  0.819488  0.680039  0.808643  0.821004  0.680015
LR_LETTER_p1    0.790533  0.831482  0.244348  0.742333  0.780580  0.234162
LR_LETTER_p2    0.727400  0.727417  0.731499  0.729000  0.729119  0.731524
RF_COV          0.823509  0.824244  0.825041  0.818561  0.818820  0.816781
RF_ADULT        0.852400  0.766073  0.662463  0.854251  0.764554  0.663201
RF_LETTER_p1    0.988333  0.856021  0.820513  0.987667  0.862030  0.819864
RF_LETTER_p2    0.946200  0.946216  0.946218  0.944400  0.944337  0.943527
KNN_COV         0.777366  0.777687  0.775944  0.775262  0.775255  0.770822
KNN_ADULT       0.837778  0.742330  0.622732  0.837705  0.743980  0.625596
KNN_LETTER_p1   0.989333  0.935917  0.864636  0.990067  0.954993  0.872322
KNN_LETTER_p2   0.955600  0.955601  0.955641  0.948867  0.948872  0.948781

                    6         7         8         9         10        11  \
col
LR_COV          0.756198  0.754912  0.758021  0.747785  0.748572  0.751306
LR_ADULT        0.809405  0.818025  0.677829  0.810239  0.819795  0.681020
LR_LETTER_p1    0.779800  0.839912  0.233820  0.719067  0.794362  0.192101
LR_LETTER_p2    0.726800  0.727241  0.731490  0.721867  0.721912  0.720899
RF_COV          0.824988  0.825250  0.823244  0.818434  0.818583  0.815810
RF_ADULT        0.849389  0.771348  0.664674  0.849389  0.771186  0.664294
RF_LETTER_p1    0.991333  0.898391  0.870000  0.988067  0.863617  0.818642
RF_LETTER_p2    0.947800  0.947831  0.947692  0.946533  0.946519  0.946138
KNN_COV         0.786643  0.786982  0.785378  0.774666  0.775488  0.777683
KNN_ADULT       0.836980  0.743519  0.623922  0.836145  0.742194  0.623604
KNN_LETTER_p1   0.989867  0.950720  0.873544  0.991333  0.932341  0.882671
KNN_LETTER_p2   0.953733  0.953729  0.953297  0.953267  0.953251  0.952721

                    12        13        14
col
LR_COV          0.749609  0.750519  0.754051
LR_ADULT        0.810783  0.821328  0.681566
LR_LETTER_p1    0.795000  0.833993  0.239050
LR_LETTER_p2    0.730067  0.730266  0.733496
RF_COV          0.822344  0.822822  0.822073
```

```
RF_ADULT        0.852074   0.766555   0.661856
RF_LETTER_p1    0.989400   0.878196   0.840841
RF_LETTER_p2    0.945200   0.945315   0.945105
KNN_COV         0.776675   0.777372   0.778596
KNN_ADULT       0.834803   0.737004   0.615813
KNN_LETTER_p1   0.988867   0.923730   0.848044
KNN_LETTER_p2   0.951800   0.951822   0.951726
```

Using t_test_rel

In [62]:

```
# COV_type RF
print('For COV_type')
print('RF vs LR: ', stats.ttest_rel(df_3.loc['RF_COV'], df_3.loc['LR_COV']))

print('RF vs KNN: ', stats.ttest_rel(df_3.loc['RF_COV'], df_3.loc['KNN_COV'])

print()

print('For ADULT')

print('LR vs RF: ', stats.ttest_rel(df_3.loc['LR_ADULT'], df_3.loc['RF_ADULT'

print('LR vs KNN: ', stats.ttest_rel(df_3.loc['LR_ADULT'], df_3.loc['KNN_ADUL

print()

print('For LETTER.p1')

print('KNN vs LR: ', stats.ttest_rel(df_3.loc['KNN_LETTER_p1'], df_3.loc['LR_

print('KNN vs RF: ', stats.ttest_rel(df_3.loc['KNN_LETTER_p1'], df_3.loc['RF_

print()

print('For LETTER.p2')

print('KNN vs LR: ', stats.ttest_rel(df_3.loc['KNN_LETTER_p2'], df_3.loc['LR_

print('KNN vs RF: ', stats.ttest_rel(df_3.loc['KNN_LETTER_p2'], df_3.loc['RF_
```

```
For COV_type
RF vs LR:  Ttest_relResult(statistic=103.28120454334169, pvalue=1.393116137558
4298e-21)
RF vs KNN:  Ttest_relResult(statistic=47.225397412572036, pvalue=7.72199642077
0533e-17)

For ADULT
LR vs RF:  Ttest_relResult(statistic=0.873826585423997, pvalue=0.3969546434237
6696)
LR vs KNN:  Ttest_relResult(statistic=2.9851246514486975, pvalue=0.00983726711
8014317)

For LETTER.p1
KNN vs LR:  Ttest_relResult(statistic=5.46859010572312, pvalue=8.2755349153246
47e-05)
KNN vs RF:  Ttest_relResult(statistic=3.9707463837257824, pvalue=0.00139379741
15811253)

For LETTER.p2
KNN vs LR:  Ttest_relResult(statistic=175.3454216353315, pvalue=8.476256645607
019e-25)
KNN vs RF:  Ttest_relResult(statistic=16.021705764895174, pvalue=2.12189135687
78697e-10)
```

In [65]:

```
print('Using t_test ind')

print('For COV_type')
```

```
print('RF vs LR: ', stats.ttest_ind(df_3.loc['RF_COV'], df_3.loc['LR_COV']))

print('RF vs KNN: ', stats.ttest_ind(df_3.loc['RF_COV'], df_3.loc['KNN_COV'])

print()

print('For ADULT')

print('LR vs RF: ', stats.ttest_ind(df_3.loc['LR_ADULT'], df_3.loc['RF_ADULT'

print('LR vs KNN: ', stats.ttest_ind(df_3.loc['LR_ADULT'], df_3.loc['KNN_ADUL'

print()

print('For LETTER.p1')

print('KNN vs LR: ', stats.ttest_ind(df_3.loc['KNN_LETTER_p1'], df_3.loc['LR_

print('KNN vs RF: ', stats.ttest_ind(df_3.loc['KNN_LETTER_p1'], df_3.loc['RF_

print()

print('For LETTER.p2')

print('KNN vs LR: ', stats.ttest_ind(df_3.loc['KNN_LETTER_p2'], df_3.loc['LR_

print('KNN vs RF: ', stats.ttest_ind(df_3.loc['KNN_LETTER_p2'], df_3.loc['RF_
```

```
Using t_test ind
For COV_type
RF vs LR:  Ttest_indResult(statistic=61.805301937963485, pvalue=1.750052156455
044e-31)
RF vs KNN:  Ttest_indResult(statistic=29.70274115306474, pvalue=1.029803358259
5094e-22)

For ADULT
LR vs RF:  Ttest_indResult(statistic=0.338542692769459, pvalue=0.7374798647637
011)
LR vs KNN:  Ttest_indResult(statistic=1.2545249184028562, pvalue=0.22002283565
525996)

For LETTER.p1
KNN vs LR:  Ttest_indResult(statistic=4.5340049117333185, pvalue=9.90424931924
8744e-05)
KNN vs RF:  Ttest_indResult(statistic=1.5211009575385597, pvalue=0.13944835217
102505)

For LETTER.p2
KNN vs LR:  Ttest_indResult(statistic=193.75960937023683, pvalue=2.43749540709
08364e-45)
KNN vs RF:  Ttest_indResult(statistic=9.675551620076511, pvalue=1.983343034677
2277e-10)
```

Conducting t-test analysis on Training data

In [69]:

```
LR_acc_tr = [0.760, 0.749, 0.759, 0.753, 0.754, 0.822, 0.826, 0.820,0.824 ,0.
             ,0.778,0.719 ,0.791,0.737,0.733,0.725 ,0.729,0.738]

RF_acc_tr = [1] * 20

KNN_acc_tr = [1] * 19 + [0.838]

LR_roc_tr = [0.759, 0.749, 0.756, 0.754, 0.755, 0.834, 0.828, 0.832, 0.839,0.
             0.852, 0.795, 0.831, 0.806, 0.851, 0.738, 0.733, 0.726, 0.729, 0

RF_roc_tr = [1] * 20
```

```
KNN_roc_tr = [1] * 19 + [0.746]


LR_fsc_tr = [0.766, 0.747, 0.763, 0.757, 0.756, 0.699, 0.696, 0.699, 0.699, 0
             0.237, 0.239, 0.189, 0.260, 0.738,0.738,0.724,0.734, 0.740
]

RF_fsc_tr = [1]  * 20

KNN_fsc_tr = [1] * 19 + [0.632]

print('Using t_test_ind')
# RF - highest ACC
print('For ACC:')
print('RF vs LR acc')
print(stats.ttest_ind(RF_acc_tr, LR_acc_tr))

print('RF vs KNN acc')
print(stats.ttest_ind(RF_acc_tr, KNN_acc_tr))

print('For ROC')
print('RF vs LR roc')
print(stats.ttest_ind(RF_roc_tr, LR_roc_tr))

print('RF vs KNN roc')
print(stats.ttest_ind(RF_roc_tr, KNN_roc_tr))


print('For FSC')
print('RF vs LR fsc')
print(stats.ttest_ind(RF_fsc_tr, LR_fsc_tr))

print('RF vs KNN fsc')
print(stats.ttest_ind(RF_fsc_tr, KNN_fsc_tr))
```

```
Using t_test_ind
For ACC:
RF vs LR acc
Ttest_indResult(statistic=28.69762473400603, pvalue=2.3121231873332913e-27)
RF vs KNN acc
Ttest_indResult(statistic=0.9999999999999993, pvalue=0.32363608386440845)
For ROC
RF vs LR roc
Ttest_indResult(statistic=20.66017392411133, pvalue=2.909520461782259e-22)
RF vs KNN roc
Ttest_indResult(statistic=1.0000000000000033, pvalue=0.32363608386440657)
For FSC
RF vs LR fsc
Ttest_indResult(statistic=7.935692168622103, pvalue=1.384168450927912e-09)
RF vs KNN fsc
Ttest_indResult(statistic=0.9999999999999983, pvalue=0.323636083864409)
```

Table 3 training Values p-test comparisons

In [71]:

```
LR_COV = [
0.760,
0.749,
0.759,
0.753,
0.754,
0.759,
0.749,
0.756,
0.754,
0.755,
0.766,
```

```
0.747,
0.763,
0.757,
0.756
]

RF_COV = [1] * 15
KNN_COV = [1] * 15


LR_ADULT = [ 0.822, 0.826, 0.820, 0.824, 0.808, 0.834, 0.828, 0.832,
             0.839, 0.823, 0.699, 0.696, 0.699, 0.699, 0.682]

RF_ADULT = [1] * 15
KNN_ADULT = [
1.000,
1.000,
0.838,
1.000,
1.000,
1.000,
1.000,
0.746,
1.000,
1.000,
1.000,
1.000,
0.632,
1.000,
1.000
]

LR_L1 = [
0.789,
0.755,
0.778,
0.719,
0.791,
0.852,
0.795,
0.831,
0.806,
0.851,
0.232,
0.237,
0.239,
0.189,
0.260
]


RF_L1 = [1] * 15

KNN_L1 = [1] * 15

LR_L2 = [
0.737,
0.733,
0.725,
0.729,
0.738,
0.738,
0.733,
0.726,
0.729,
```

```
0.739,
0.738,
0.738,
0.724,
0.734,
0.740
]

RF_L2 = [1] * 15
KNN_L2 = [1] * 15
```

In [72]:

```
print('Using t_test ind')

print('For COV_type')
print('RF vs LR: ', stats.ttest_ind(RF_COV, LR_COV))

print('RF vs KNN: ', stats.ttest_ind(RF_COV, KNN_COV))

print()

print('For ADULT')

print('RF vs LR: ', stats.ttest_ind(RF_ADULT, LR_ADULT))

print('RF vs KNN: ', stats.ttest_ind(RF_ADULT, KNN_ADULT))

print()

print('For LETTER.p1')

print('RF vs LR: ', stats.ttest_ind(RF_L1, LR_L1))

print('RF vs KNN: ', stats.ttest_ind(RF_L1, KNN_L1))

print()

print('For LETTER.p2')

print('RF vs LR: ', stats.ttest_ind(RF_L2, LR_L2))

print('RF vs KNN: ', stats.ttest_ind(RF_L2, KNN_L2))
```

```
Using t_test ind
For COV_type
RF vs LR:  Ttest_indResult(statistic=181.44084209408493, pvalue=1.531504895746
1175e-44)
RF vs KNN:  Ttest_indResult(statistic=nan, pvalue=nan)

For ADULT
RF vs LR:  Ttest_indResult(statistic=13.143816790213068, pvalue=1.688565382346
1946e-13)
RF vs KNN:  Ttest_indResult(statistic=1.7599722921995125, pvalue=0.08933491376
03266)

For LETTER.p1
RF vs LR:  Ttest_indResult(statistic=5.453859797011504, pvalue=8.0322807117706
11e-06)
RF vs KNN:  Ttest_indResult(statistic=nan, pvalue=nan)

For LETTER.p2
RF vs LR:  Ttest_indResult(statistic=187.27033302575958, pvalue=6.322290914528
812e-45)
RF vs KNN:  Ttest_indResult(statistic=nan, pvalue=nan)
```

In [ ]:

Table 2:				
MODEL	ACC	FSC	ROC	MEAN
LR	0.7632978486	0.5980563747	0.7786242399	0.7133261544
RF	**0.9020135585**	**0.815898878**	*0.851895492237036	0.8589562182
KNN	*0.889337816468553	*0.805173799268046	**0.8531394655**	0.8531394655
* - corresponding to t_test_ind p values				
Table 6: P-vals for T	using t_test rel			
MODEL	ACC	FSC	ROC	
LR	8.46E-07	0.000815166617	0.01005296938	
RF	1	1	0.901906609	
KNN	0.01072652824	0.1976641606	1	
Table 6: P vals for Table 2 using t_test ind				
	ACC	FSC	ROC	
LR	1.28E-09	0.00028676665	0.003192436454	
RF	1	1	0.9626539677	
KNN	0.6162507401	0.770120787	1	

MODEL	COV_type	ADULT	LETTER.p1	LETTER.p2	MEAN
LR	0.7519790943	**0.7699562466**	0.6033696005	0.7279996761	0.7133261544
RF	**0.8213670646**	*0.760913738192151	*0.898194312649595	0.9459354562	0.8836512604
KNN	0.7781212999	*0.733607078998716	**0.9325590798**	**0.9525806496**	0.8877536764
*- corresponding to t_test_ind values					
Table 7: Table 3 p-test using t_test_rel					
MODEL	COV_type	ADULT	LETTER.p1	LETTER.p2	
LR	1.39E-21	1	8.28E-05	8.48E-25	
RF	1	0.3969546434	0.001393797412	2.12E-10	
KNN	7.72E-17	0.009837267118	1	1	
Table 7: Table 3 p-test using t_test_ind					
MODEL	COV_type	ADULT	LETTER.p1	LETTER.p2	
LR	1.75E-31	1	9.90E-05	2.44E-45	
RF	1	0.7374798648	0.1394483522	1.98E-10	
KNN	1.03E-22	0.2200228357	1	1	