An Empirical Comparison of Supervised Learning Algorithms Replication

Jou-Ying Lee [A14685605] jol067@ucsd.edu

Department of Data Science, University of California San Diego, CA 92093 USA

Abstract

With numbers of machine learning algorithms introduced in the past decade, this study replicates prior work done by Caruana and Niculescu-Mizil -- returns back to the fundamentals, and studies supervised learning algorithms in details. An empirical comparison of three of the most popular and commonly used supervised learning algorithms: k-nearest neighbors, logistic regression, and random forest will be presented. Prediction tasks in this analysis is structured purposefully as classification tasks, with datasets applied for modelling hand-picked for prediction outputs to be evaluated by a same performance criteria.

Keywords

k-nearest neighbors, logistic regression, random forest, accuracy score, supervised learning

I. Introduction

With data produced and collected at an exponential speed today, numbers of techniques have been introduced to capture and learn these great amount of transferable knowledge. Machine learning as a special study for automated pattern recognitions, is now more and more prevalently used as a tool for such information processing and prediction tasks.

Supervised learning as a subset of machine learning, is a special application of such automation tasks. With different functions developed for specific modeling needs, learning algorithms each come with distinct hyperparameters set -- which by altering can greatly affect prediction outcomes. It is therefore important to study and evaluate how varying machine learning models' parameters affect final predictions, and how they alter on different forms of input datasets.

To ensure balance in cross comparisons over datasets and algorithms, this study focuses on classification tasks only. Evaluation of model performances is therefore more comparable and unbiased by a same scoring criteria: the accuracy score between true labels and predicted outcomes.

Three of the most popular classification models will be implemented and studied for result comparison. These are: k-nearest neighbors, logistic regression and random forest classifier. These algorithms are carefully picked for study because they each lie within a different class of supervised learning models, namely: neighbors-based model, linear model and ensemble-based methods for classification.

By studying each of these model parameters, and comparing their performance results on different datasets, this paper hopes to provide empirical insights into supervised learning algorithms' strengths and weaknesses.

II. Methodology

A. Dataset

Comparison of the algorithms on 3 binary classification problems is carried out. This study utilizes the ADULT, BANK_MARKETING and LETTER_RECOGNITION datasets from the UCI Machine Learning Repository (Blake & Merz, 1998).

ADULT comes with a structured binary classification task, to predict whether a person's income exceeds $50k/year based on the census data (Blake & Merz, 1998). The "*education*" feature can be completely represented by the "*education_num*" column, and therefore is dropped from the dataframe before training to avoid multicollinearity. The label column is converted into integers of 0 and 1, whereas the rest of the categorical features are one-hot encoded for training purposes.

BANK_MARKETING (BM) is also built with a binary classification goal to predict if a client will subscribe to a term deposit based on this direct marketing campaign dataset of a Portuguese banking institute (Blake & Merz, 1998) Columns with binary values are converted into numerical values of 0s and 1s, and categorical feature columns are one-hot encoded before training.

LETTER_RECOGNITION (LR) is a multivariate dataset with character image features of the 26 alphabets (Blake & Merz, 1998). Different from previous two input datasets, this dataframe has 26 values in its label column originally. This prediction task is therefore transformed to a binary labeled problem by treating letters *A-M* as label 0, and letters *N-Z* as label 1.

B. Learning Algorithms

This replication attempts to "explore the space of parameters and common variations for each algorithm" (Caruana and Niculescu-Mizil). This section summarizes the hyper-parameter sets implemented for training for each learning algorithm.

K-Nearest Neighbors (KNN) is a neighbors-based supervised learning algorithm. It is a type of instance-based learning and therefore stores all training instances in memory for training. Tuned hyper-parameters for this model include *k*: specifying the number of neighbors for queries, and the *weight* function used in prediction. 25 values of *k* ranging from 1-500 spaced evenly on a log scale is implemented for training. *Distance weighted* and *uniformly weighted* KNNs are both cross validated with different *k* values during training.

Logistic Regression (LOG) is a linear model which leverages the logit function for describing probabilities of outcomes in each trial for classification. Both regularized and unregularized models are trained, with the ridge regularization parameter applied by factors of 10 from 10^{-8} to 10^{4}.

Random Forest (RF) is an ensemble method based on randomized decision trees. Breiman-Cutler's implementation is employed (implementation executed by the scikit-learn library). The classifier is fit with 1024 decision trees, with the size of the features to be considered for "best split" set at: 1, 2, 4, 6, 8, 12, 16 or 20.

In order to speed up model training time, all attributes from the datasets are scaled to 0 mean and 1 standard deviation before fitted to the algorithms above.

C. Performance Metric

All algorithm/dataset performances are measured by a same performance metric: the accuracy classification score. The predicted class for a sample must exactly match the corresponding true label for an accuracy score of 100%; Otherwise, accuracy scores are measured by taking the percentage of correct predictions out of the total number of predictions made (i.e. the number of entries in each dataset).

III. EXPERIMENT

For each test problem, 5000 cases are randomly selected for training, and the rest of the cases are used as a large final test set. 3-fold cross validation is applied to each of the 5000 cases to obtain 3 trials. During each trial, about 3300 cases are used for training the different models, and the rest of about 1700 cases are implemented for use of calibrating the models and selecting the best-performing hyper-parameters set. Test set performance is then finally reported on the remaining large dataset.

A. Performance by Metric

Table I shows the accuracy scores for each of the algorithms on the accuracy score metric. The best parameter settings for each algorithm is found using the ~1700 validation entries set aside by cross validation. The model's accuracy score is then reported on the final test set. Each entry in Table I is an average of these scores across the three trials and three classication problems. The model with the best performance is marked in **bold**, and algorithms whose performance is not statistically distinguishable from the best algorithm at $p = 0.05$ using two-sided sample t-tests on the three trials are marked with "*". Entry in the table that is neither bold nor starred indicates performance that is *siginificantly lower* than the best model at $p = 0.05$.

Table I. Accuracy Scores for Each Learning Algorithm (Average over 3 problems)

Model	Accuracy Score
KNN	0.884*
LOG	0.824
RF	**0.904**

Random Forest turns out to have the best test-set performance out of the three algorithms.

To compare across algorithms, two sample t-tests are carried out to determine whether or not the difference in mean accuracies between the best performing algorithm (i.e. Random Forest) and the rest of the two models are statistically significant by a 95% confidence interval. It turns out that the Logistic Regression produces a measurement that is statistically different from the Random Forest depending on the difference in between average accuracy scores (p-value = 0.0098, Cohen's d =1.3824). K-nearest neighbors on the other hand, shows comparably statistically insignificant results (p-value = 0.3088, Cohen's d = 0.4956).

B. Performance by Problem

Table II shows the accuracy scores for each algorithm across each of the 3 test problems. Each entry is an average over the 3 trials when selection is done using the ~1700 validation cases.

Table II. Average Accuracy Score for Each Learning Algorithm by Problem

Model	Dataset		
	ADULT	**BM**	**LR**
KNN	0.822	0.890	**0.940**
LOG	0.847	**0.901**	0.725
RF	**0.883**	0.899	0.931

Each algorithm turns out to each have a best test-set performance across the problem instances.

Random forest, logistic regression and k-nearest neighbors each has a highest average accuracy score of 0.883, 0.901 and 0.94 throughout the 3 trials validation on ADULT, BANK_MARKETING and LETTER_RECOGNITION.

Table III is a summary of the statistical results of performed two sample t-tests. Mean accuracies between the best performing algorithm on each dataset is compared with the rest of the models. P-values and Cohen's d value are both reported in Table III. At $p = 0.05$, none of the algorithms at none of the classification problems showed statistically insignificant results judging by their *p-values* significantly below 0.05. This is also reflected in Table II, where none of the entries are marked with "*".

Table III. Two Sample t-test Summary by Problem

Model	ADULT		BM		LR	
	RF		LOG		KNN	
	P	*D*	*P*	*D*	*P*	*D*
KNN	.018	3.157	0	25.06	/	/
LOG	.083	1.877	/	/	0	172.6
RF	/	/	.0662	2.048	.0004	8.78

C. Performance by Optimal Hyper-Parameters Set

Hyper-parameters for each algorithm on each problem dataset is selected via each 3 cross validations. By training each model with its best parameter set, we obtain the "optimal training accuracy" and "optimal testing accuracy" by fitting the model to the entire training and testing set accordingly.

Table IV shows the mean *training set* as well as the *mean testing set* performance for the optimal hyperparameters on each of the dataset/algorithm combo. Under each of the "dataset", cell to the left represents "optimal training

accuracy, while that on the right refers to the "optimal testing accuracy".

Table IV. Average Training Accuracy Score for Each Optimized Learning Algorithm by Problem

Model	Dataset					
	ADULT		BM		LR	
KNN	**1**	0.825	**1**	0.891	**1**	**0.954**
LOG	0.853	**0.859**	0.9	**0.901**	0.73	0.725
RF	**1**	0.857	**1**	0.9	**1**	0.947

K-nearest neighbors and random forest both obtain 100% training set accuracies on all of the problems with their corresponding optimal hyper-parameters choice. Not only on training performance, these two algorithms seem to have very alike performances on test sets as well. Reviewing outcomes of logistic regression, while it seems to suggest that logistic regression does not have as nice of a performance compared to k-nearest neighbors or random forest, pairwise comparison between each dataset's training and testing accuracy scores however reveals otherwise. Training and testing set accuracies for logistic regression across all 3 problems are all comparably close values. Logistic regression even has higher testing performance than training accuracy in two out of the three problems. This suggests an intendency for logistic regression to overfit, and is contributed by the effect of the ridge regularization term implemented for avoidance of overfitting.

IV. CONCLUSION

Research breakthroughs in this field of machine learning have been growing and revolutionizing throughout the past decade. Where more and more algorithms are being developed and implemented for all different applications today, study on model tuning has always been at the heart for artificial intelligence. Supervised learning being one of the core subset of the area, serves as an essential building block for lots of later establishments. It is therefore crucial to study in-detail these algorithms' architecture and characteristics.

This replication work compares performances of supervised learning algorithms with different hyper-parameter settings on various classification problems. By executing such empirical comparison in between each algorithm and dataset, critical insights are presented through accuracy evaluations on both models and classification problems.

K-nearest neighbors, logistic regression and random forest are calibrated with multiple hyper-parameter tuning trials, cross validated on validation folds and final unseen test datasets. While all three models present satisfying accuracy percentages on all three datasets, by comparison, conclusions in regards to each of these models' architectural nature can be drawn.

K-nearest neighbors being an instance-based algorithm is considerably slow in training. While its final prediction performance is satisfying, the model tends to overfit easily because of its memory-based nature. Large amounts of tuning and hyper-parameter try-out might be needed, especially on large datasets, before a desired accuracy level can be achieved.

Logistic regression as a linear classification model does not have as nice of a performance in comparison to the two other models judging by training and testing mean accuracies of best trained model. However, relying on its regularization power, the model has pretty nice generalization ability to avoid overfitting.

Random Forest is the only ensembling method tried out for this project, but its nature of randomness is also presented through training set and testing set performance comparison. It does not overfit easily, and in fact has the highest average testing accuracies out of the three algorithms of each with three training trials.

This replication project mainly differs from the original by implementing less algorithms, datasets and metrics for study and comparison. This therefore results in a great amount of reduction in comparison abilities across datasets and algorithms. With the lack of training power, this work also implements less cross validation trials, and may thus serve as another drawback for not being to obtain a more accurate representation of each implemented algorithm. With better computing power, this project many be extended to include additional models Caruana and Niculescu-Mizil did not include in their paper, and may be further developed for implementations of unsupervised models as well to provide more important insights into different machine learning models.

V. APPENDIX

There are content advised to report under Appendix reported elsewhere in this report, such as: the table with the p-values of the comparisons across algorithms along with t-test statistic results. This is reported in Table III, and embedded in main body report for evaluation use.

The raw test set scores (Table V) are reported here under Appendix.

Table V. (Appendix) Raw Test Accuracy Score

Model	Dataset		
	KNN	LOG	RF
ADULT	0.821	0.847	0.851
	0.821	0.846	0.900
	0.823	0.847	0.900
BM	0.890	0.901	0.900
	0.890	0.901	0.900
	0.890	0.902	0.899
LR	0.941	0.724	0.931
	0.941	0.424	0.929
	0.942	0.727	0.932

VI. BONUS

The following summarizes work I believe deserves extra bonus points:

- The paper is carefully organized and written in special details.

- Discussions on drawbacks and future directions of this work is presented.
- Evaluations on each implemented model is carried out.
- Coding portion for this project is clearly organized with outputs prettily print for clearness with reviewing and grading.

VIL. ACKNOWLEDGEMENT

[1] Blake, C., & Merz, C. (1998). UCI repository of machine learning databases.

[2] Caruana, Rich, & Niculescu-Mizil, Alexandru, "An Empirical Comparison of Supervised Learning Algorithms," ICML '06: Proceedings of the 23rd international conference on Machine learning, doi: 10.1145/1143844.1143865

[3] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011

Dependencies & Imports

In [292]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [293]:

```
import time
from bioinfokit.analys import stat
from scipy import stats
```

In [294]:

```
import pandas as pd
import string
import numpy as np
```

In [295]:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler, MinMaxScaler
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import GridSearchCV, StratifiedKFold, train_test_split
from sklearn.metrics import classification_report, accuracy_score, f1_score
```

In [296]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier as Knn
from sklearn.ensemble import RandomForestClassifier
```

Read in Data

In [297]:

```
# list column names

adult_col_name = ['age',
                  'workclass',
                  'fnlwgt',
                 'education',
                 'education_num',
                 'marital_status',
                  'occupation',
                  'relationship',
                  'race',
                  'sex',
                  'capital-gain',
                  'capital-loss',
                  'hours-per-week',
                  'native-country',
                  'label_50k'
                 ]

letter_col_name = [
    'letter',
    'x_box',
    'y_box',
    'width',
    'height',
    'onpix', # total number on pixels
    'x_bar', # mean x of on pixels in box
    'y_bar', # mean y of on pixels in box
    'x2bar', # mean x variance (integer)
    'y2bar', # mean y variance (integer)
    'xybar', # mean x y correlation (integer)
    'x2ybr', # mean of x * x * y (integer)
    'xy2br', # mean of x * y * y (integer)
    'x_ege', # mean edge count left to right (integer)
    'xegvy', # correlation of x-ege with y (integer)
    'y_ege', # mean edge count bottom to top (integer)
    'yegvx', # correlation of y-ege with x (integer)
]
```

In [298]:

```
# Adult
adult_df = pd.read_csv('data/adult.data', names=adult_col_name)
# Bank Marketing
bank_df = pd.read_csv('data/bank-full.csv', sep=';', header=0)
# Letter Recognition
letter_df = pd.read_csv('data/letter-recognition.data', names=letter_col_name)
```

Check missing values.

In [299]:

```
adult_df.isnull().sum()
print('No missing values in adult data')

bank_df.isnull().sum()
print('No missing values in bank marketing data')

letter_df.isnull().sum()
print('No missing values in letter data')
```

```
No missing values in adult data
No missing values in bank marketing data
No missing values in letter data
```

Feature Engineering

In [300]:

```
def adult_feat_eng(adult_df):

    # `education` dropped because can be represented by education_num
    adult_df.drop('education', axis=1, inplace=True)

    # transform label column to 0, 1
    adult_df.label_50k.replace({' <=50K':0, ' >50K':1}, inplace=True)

    # one-hot encode
    cat_cols = adult_df.select_dtypes('object').columns
    adult_df = pd.get_dummies(adult_df, columns=cat_cols, drop_first=True)


    return adult_df
```

In [301]:

```
def bank_feat_eng(bank_df):

    # transform columns to 0, 1
    yes_no_cols = ['default', 'housing', 'loan', 'y']

    for col in yes_no_cols:
        bank_df[col].replace({'no':0, 'yes':1}, inplace=True)

    cat_cols = bank_df.select_dtypes('object').columns
    bank_df = pd.get_dummies(bank_df, columns=cat_cols, drop_first=True)

    return bank_df
```

In [302]:

```
def letter_feat_eng(letter_df):

    # transform label column to 0, 1
    letter_upper = string.ascii_uppercase
    positive = {i:0 for i in letter_upper[:(letter_upper.index('M')+1)]}
    negative = {i:1 for i in letter_upper[(letter_upper.index('M')+1):]}
    positive.update(negative)
    # replace letters (labels) as LETTER.p2 in CNM06
    # A-M = 0
    # N-Z = 1
    letter_df.letter.replace(positive, inplace=True)

    return letter_df
```

Fit, Transform datasets

Adult

In [303]:

```
# feature engineer datasets
adult_df = adult_feat_eng(adult_df)
```

In [304]:

```
adult_X = adult_df.drop('label_50k', axis=1)
adult_y = adult_df.label_50k
```

Bank

In [305]:

```
bank_df = bank_feat_eng(bank_df)
```

In [306]:

```
bank_X = bank_df.drop('y', axis=1)
bank_y = bank_df.y
```

Letter

In [307]:

```
letter_df = letter_feat_eng(letter_df)
```

In [308]:

```
letter_X = letter_df.drop('letter', axis=1)
letter_y = letter_df.letter
```

Modeling

In [309]:

```
# Define classification models
models = {
    'KNN': Knn(),
    'Logistic_Regression': LogisticRegression(penalty='l2', random_state=42),
    'Random_Forest': RandomForestClassifier(n_estimators=1024, random_state=42)
}
```

In [310]:

```
# Define model hyperparameters grid
params = {
    'KNN':{
        # 25 numbers from 1-500, spaced evenly on a log scale
        'classifier__n_neighbors': np.geomspace(1, 500, num=25, dtype='int'),
        # 'uniform': default, 'distance': disstance weighting
        'classifier__weights': ['uniform', 'distance']

    },
    'Logistic_Regression': {
        'classifier__C': [(10**(-8+i)) for i in range(0, 13)]+[0] #10^-8 to 10^4 and
    },
    'Random_Forest': {
        'classifier__max_features': [1, 2, 4, 6, 8, 12, 16, 20]
    }
}
```

In [311]:

```
def pipeline(model):

    pipe = Pipeline([('std', StandardScaler()), ('classifier', model)])

    return pipe
```

In [312]:

```
def grid_cvs_objects(models, params):

    grid_cvs = {}

    for model_name in models:

        model = models[model_name]
        pipe = pipeline(model)
        param_grid = params[model_name]

        clf = GridSearchCV(
            estimator = pipe,
            param_grid = param_grid,
            n_jobs = 1,
            cv = 2,
            verbose = 0,
            refit = True
        )
        grid_cvs[model_name] = clf

    return grid_cvs
```

In [313]:

```
# Define GridCV for each model
grid_cvs = grid_cvs_objects(models, params)
```

Model Performance Evaluation

In [315]:

```
# Outer Fold 1/3 | tuning KNN| inner ACC 82.00% | outer ACC 100.00%
def model_accuracy(X_train, y_train, X_test, y_test, grid_cvs):

    start_time = time.time()

    cv_scores = {model: [] for model in grid_cvs.keys()}
    s_kfold = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
    c = 1

    # outer loop: model selection
    for outer_train_idx, outer_valid_idx in s_kfold.split(X_train,y_train):

        a, b = 3, 4
        for name, gs_est in sorted(grid_cvs.items()):

            print('outer fold %d/3 | tuning %-8s' % (c, name), end='')

            # The inner loop for hyperparameter tuning
            gs_est.fit(X_train.iloc[outer_train_idx], y_train.iloc[outer_train_idx])
            y_pred = gs_est.predict(X_train.iloc[outer_valid_idx])
            acc = accuracy_score(y_true=y_train.iloc[outer_valid_idx], y_pred=y_pred
            print(' | inner ACC %.2f%% | outer ACC %.2f%%' %
                  (gs_est.best_score_ * 100, acc * 100))
            cv_scores[name].append(acc)

            y_testset_pred = gs_est.predict(X_test)
            acc_test = accuracy_score(y_true=y_test, y_pred=y_testset_pred)
            print(' | final test set ACC {} **\n'.format(acc_test*100))

        c += 1

    end_time = time.time()
    print('\n== Elapsed Time:', (end_time - start_time),'==')

    return cv_scores
```

In [316]:

```
def model_result(cv_scores, grid_cvs):

    # Looking at the results
    for name in cv_scores:
        print('%-8s | outer CV acc. %.2f%% +\- %.3f' % (
              name, 100 * np.mean(cv_scores[name]), 100 * np.std(cv_scores[name])))

    print()
    for name in cv_scores:
        print('{} best parameters'.format(name), grid_cvs[name].best_params_)
```

In [317]:

```
def model_entire_df(grid_cvs, X_train, X_test, y_train, y_test):

    # Fitting a model to the whole training set
    # using the "best" algorithm

    for model in grid_cvs:
        best_algo = grid_cvs[model]

        best_algo.fit(X_train, y_train)
        train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train)
        test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))

        print(model)
        print('Accuracy %.2f%% (average over CV test folds)' %
              (100 * best_algo.best_score_))
        print('Best Parameters: %s' % best_algo.best_params_)
        print('Training Accuracy: %.2f%%' % (100 * train_acc))
        print('Test Accuracy: %.2f%%' % (100 * test_acc))
        print()
```

In [318]:

```
# model_performance for 1 dataset
def model_performance(X, y, grid_cvs):

    X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                        train_size=5000,
                                                        random_state=42,
                                                        stratify=y)

    # model_accuracy of 1 dataset 3 models
    print('Model Accuracy:\n')
    cv_scores = model_accuracy(X_train, y_train, X_test, y_test, grid_cvs)
    print()

    # average score & best params of each model
    print('Model Result (Average Score & Best Params for Each Model): \n')
    model_result(cv_scores, grid_cvs)
    print()

    # Fit best model to the whole set
    print('Model: (Average Score) on Whole Set:')
    print()
    model_entire_df(grid_cvs, X_train, X_test, y_train, y_test)
```

In [319]:

```
def main_caller():

    for dataset_name, X, y in zip(
        ('Adult_df', 'Bank_df', 'Letter_df'),
        (adult_X, bank_X, letter_X),
        (adult_y, bank_y, letter_y)
                                  ):
        print('**Model Performance for {}**:\n'.format(dataset_name))
        model_performance(X, y, grid_cvs)
        print()
        print('---------------------------------')
        print()
```

In [336]:

```
# take in average scores of models on each dataset
def two_sample_t_test_model(test_alg_acc, best_performance_model):

    best_model_acc = test_alg_acc[best_performance_model]
    for i in test_alg_acc:
        if i != best_performance_model:
            nxt_model_acc = test_alg_acc[i]
            df_dict = {best_performance_model: best_model_acc,
                       i: nxt_model_acc}
            df = pd.DataFrame(df_dict)
            print('t-test stats for {} vs {}:'.format(best_performance_model, i))
            print()
            stats = researchpy.ttest(df[best_performance_model], df[i])[1]
            print('t =', stats.results[2])
            print('Two side test p value =', stats.results[3])
            print("Cohen's d =", stats.results[6])
            print()
            print('-----------------------------------------------------------')
            print()
```

In [359]:

```
def two_sample_t_test_dataset(test_alg_acc, df_name, best_perform_name):
    x = 0
    for i, j in zip(df_name, best_perform_name):
        print('**{}**\n'.format(i))
        best_model_acc = test_alg_acc[j][x:x+3]
        for k in ['rf', 'log', 'knn']:
            if k != j:
                nxt_model_acc = test_alg_acc[k][x:x+3]
                df_dict = {j: best_model_acc,
                           k: nxt_model_acc}
                df = pd.DataFrame(df_dict)
                print('t-test stats for {} vs {}:'.format(j, k))
                print()
                stats = researchpy.ttest(df[j], df[k])[1]
                print('t =', stats.results[2])
                print('Two side test p value =', stats.results[3])
                print("Cohen's d =", stats.results[6])
                print()
                print('------------------------------------------------------------')
                print()
        x+=3
```

In [320]:

```
main_caller()
```

```
**Model Performance for Adult_df**:

Model Accuracy:

outer fold 1/3 | tuning KNN      | inner ACC 82.00% | outer ACC 82.42%
 | final test set ACC 82.07249374115598 **

outer fold 1/3 | tuning Logistic_Regression | inner ACC 84.01% | outer
ACC 83.92%
 | final test set ACC 84.72116396357171 **

outer fold 1/3 | tuning Random_Forest | inner ACC 84.61% | outer ACC 8
4.34%
 | final test set ACC 85.13116360074018 **

outer fold 2/3 | tuning KNN      | inner ACC 82.36% | outer ACC 82.36%
 | final test set ACC 82.09063531802184 **

outer fold 2/3 | tuning Logistic_Regression | inner ACC 83.83% | outer
ACC 84.46%
 | final test set ACC 84.5651464025253 **

outer fold 2/3 | tuning Random_Forest | inner ACC 84.13% | outer ACC 8
4.76%
 | final test set ACC 85.67903922208919 **

outer fold 3/3 | tuning KNN      | inner ACC 82.66% | outer ACC 82.35%
 | final test set ACC 82.3301041326512 **

outer fold 3/3 | tuning Logistic_Regression | inner ACC 83.89% | outer
ACC 84.57%
 | final test set ACC 84.67762417909364 **

outer fold 3/3 | tuning Random_Forest | inner ACC 85.00% | outer ACC 8
4.99%
 | final test set ACC 85.2472696926817 **


== Elapsed Time: 364.14927792549133 ==

Model Result (Average Score & Best Params for Each Model):

KNN      | outer CV acc. 82.38% +\- 0.031
Logistic_Regression | outer CV acc. 84.32% +\- 0.284
Random_Forest | outer CV acc. 84.70% +\- 0.269

KNN best parameters {'classifier__n_neighbors': 62, 'classifier__weigh
ts': 'distance'}
Logistic_Regression best parameters {'classifier__C': 0.1}
Random_Forest best parameters {'classifier__max_features': 12}

Model: (Average Score) on Whole Set:

KNN
Accuracy 82.28% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 105, 'classifier__weight
s': 'distance'}
```

```
Training Accuracy: 100.00%
Test Accuracy: 82.46%

Logistic_Regression
Accuracy 84.06% (average over CV test folds)
Best Parameters: {'classifier__C': 1}
Training Accuracy: 85.30%
Test Accuracy: 84.88%

Random_Forest
Accuracy 84.70% (average over CV test folds)
Best Parameters: {'classifier__max_features': 12}
Training Accuracy: 100.00%
Test Accuracy: 85.65%


--------------------------------

**Model Performance for Bank_df**:

Model Accuracy:

outer fold 1/3 | tuning KNN      | inner ACC 88.63% | outer ACC 89.80%
 | final test set ACC 88.99803536345776 **

outer fold 1/3 | tuning Logistic_Regression | inner ACC 89.41% | outer
ACC 89.14%
 | final test set ACC 90.09226331103429 **

outer fold 1/3 | tuning Random_Forest | inner ACC 89.83% | outer ACC 9
0.34%
 | final test set ACC 90.01765686006317 **

outer fold 2/3 | tuning KNN      | inner ACC 88.99% | outer ACC 89.26%
 | final test set ACC 88.9607321379722 **

outer fold 2/3 | tuning Logistic_Regression | inner ACC 89.68% | outer
ACC 89.74%
 | final test set ACC 90.05744696724777 **

outer fold 2/3 | tuning Random_Forest | inner ACC 89.59% | outer ACC 9
0.22%
 | final test set ACC 89.95797169928626 **

outer fold 3/3 | tuning KNN      | inner ACC 89.17% | outer ACC 89.38%
 | final test set ACC 89.03036482554525 **

outer fold 3/3 | tuning Logistic_Regression | inner ACC 89.44% | outer
ACC 89.62%
 | final test set ACC 90.15940911690831 **

outer fold 3/3 | tuning Random_Forest | inner ACC 89.92% | outer ACC 8
9.50%
 | final test set ACC 89.80378503394594 **


== Elapsed Time: 326.4383580684662 ==

Model Result (Average Score & Best Params for Each Model):

KNN      | outer CV acc. 89.48% +\- 0.232
```

```
Logistic_Regression | outer CV acc. 89.50% +\- 0.258
Random_Forest | outer CV acc. 90.02% +\- 0.374

KNN best parameters {'classifier__n_neighbors': 10, 'classifier__weigh
ts': 'distance'}
Logistic_Regression best parameters {'classifier__C': 1}
Random_Forest best parameters {'classifier__max_features': 16}

Model: (Average Score) on Whole Set:

KNN
Accuracy 88.90% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 28, 'classifier__weight
s': 'distance'}
Training Accuracy: 100.00%
Test Accuracy: 89.11%

Logistic_Regression
Accuracy 89.56% (average over CV test folds)
Best Parameters: {'classifier__C': 0.1}
Training Accuracy: 89.96%
Test Accuracy: 90.17%

Random_Forest
Accuracy 89.88% (average over CV test folds)
Best Parameters: {'classifier__max_features': 20}
Training Accuracy: 100.00%
Test Accuracy: 89.99%


--------------------------------

**Model Performance for Letter_df**:

Model Accuracy:

outer fold 1/3 | tuning KNN      | inner ACC 91.18% | outer ACC 94.24%
 | final test set ACC 94.05333333333333 **

outer fold 1/3 | tuning Logistic_Regression | inner ACC 72.61% | outer
ACC 72.11%
 | final test set ACC 72.41333333333333 **

outer fold 1/3 | tuning Random_Forest | inner ACC 90.61% | outer ACC 9
4.18%
 | final test set ACC 93.14666666666666 **

outer fold 2/3 | tuning KNN      | inner ACC 91.36% | outer ACC 93.58%
 | final test set ACC 94.06 **

outer fold 2/3 | tuning Logistic_Regression | inner ACC 73.42% | outer
ACC 71.99%
 | final test set ACC 72.36666666666667 **

outer fold 2/3 | tuning Random_Forest | inner ACC 91.00% | outer ACC 9
2.98%
 | final test set ACC 92.92 **

outer fold 3/3 | tuning KNN      | inner ACC 91.57% | outer ACC 94.24%
 | final test set ACC 94.16666666666667 **
```

```
outer fold 3/3 | tuning Logistic_Regression | inner ACC 73.22% | outer
ACC 73.05%
 | final test set ACC 72.67333333333333 **

outer fold 3/3 | tuning Random_Forest | inner ACC 91.18% | outer ACC 9
3.82%
 | final test set ACC 93.2 **


== Elapsed Time: 250.084725856781 ==

Model Result (Average Score & Best Params for Each Model):

KNN      | outer CV acc. 94.02% +\- 0.310
Logistic_Regression | outer CV acc. 72.38% +\- 0.476
Random_Forest | outer CV acc. 93.66% +\- 0.502

KNN best parameters {'classifier__n_neighbors': 1, 'classifier__weight
s': 'uniform'}
Logistic_Regression best parameters {'classifier__C': 10}
Random_Forest best parameters {'classifier__max_features': 8}

Model: (Average Score) on Whole Set:

KNN
Accuracy 92.90% (average over CV test folds)
Best Parameters: {'classifier__n_neighbors': 4, 'classifier__weights':
'distance'}
Training Accuracy: 100.00%
Test Accuracy: 95.41%

Logistic_Regression
Accuracy 72.84% (average over CV test folds)
Best Parameters: {'classifier__C': 0.1}
Training Accuracy: 73.00%
Test Accuracy: 72.52%

Random_Forest
Accuracy 92.46% (average over CV test folds)
Best Parameters: {'classifier__max_features': 4}
Training Accuracy: 100.00%
Test Accuracy: 94.68%


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

In [321]:

```
alg_accs = {
    # across adult, bank, letter
    'knn': np.array([82.38, 89.48, 94.02])*0.01,
    'log': np.array([84.32, 89.50, 72.38])*0.01,
    'rf': np.array([84.70, 90.02, 93.66])*0.01
}
```

In [324]:

```
test_alg_acc = {
    'knn': np.array([82.07249374115598, 82.09063531802184, 82.3301041326512,
                     88.99803536345776, 88.9607321379722, 89.03036482554525,
                      94.05333333333333, 94.06, 94.16666666666667
                     ])*0.01,
    'log': np.array([84.72116396357171, 84.5651464025253, 84.67762417909364,
                     90.09226331103429, 90.05744696724777, 90.15940911690831,
                     72.41333333333333, 72.36666666666667, 72.67333333333333
                     ])*0.01,
    'rf': np.array([85.13116360074018, 89.95797169928626, 89.80378503394594,
                    90.01765686006317, 89.95797169928626, 89.80378503394594,
                     93.14666666666666, 92.92, 93.2
                    ])*0.01
}
```

In [325]:

```
# TABLE 1
for i in test_alg_acc.keys():
    print(i, np.mean(test_alg_acc[i]))
```

```
knn 0.8841804061320047
log 0.8241404303041271
rf 0.9043766673265939
```

In [335]:

```
# TABLE 2
x = 0
for i in ('ADULT','BM','LR'):
    for j in test_alg_acc.keys():
        print(i, j, np.mean(test_alg_acc[j][x:x+3]))
    x += 3
```

```
ADULT knn 0.8216441106394301
ADULT log 0.8465464484839688
ADULT rf 0.8829764011132412
BM knn 0.8899637744232507
BM log 0.9010303979839679
BM rf 0.8992647119776512
LR knn 0.9409333333333333
LR log 0.7248444444444444
LR rf 0.930888888888889
```

In [337]:

```
two_sample_t_test_model(test_alg_acc, 'rf')
```

```
t-test stats for rf vs knn:

t = 1.0513
Two side test p value = 0.3088
Cohen's d = 0.4956

----------------------------------------------------------

t-test stats for rf vs log:

t = 2.9325
Two side test p value = 0.0098
Cohen's d = 1.3824

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

In [360]:

```
two_sample_t_test_dataset(test_alg_acc, ('ADULT','BM','LR'), ('rf', 'log', 'knn'))
```

```
**ADULT**

t-test stats for rf vs log:

t = 2.2991
Two side test p value = 0.083
Cohen's d = 1.8772

----------------------------------------------------

t-test stats for rf vs knn:

t = 3.867
Two side test p value = 0.018
Cohen's d = 3.1574

----------------------------------------------------

**BM**

t-test stats for log vs rf:

t = 2.5083
Two side test p value = 0.0662
Cohen's d = 2.048

----------------------------------------------------

t-test stats for log vs knn:

t = 30.6917
Two side test p value = 0.0
Cohen's d = 25.0597

----------------------------------------------------

**LR**

t-test stats for knn vs rf:

t = 10.7589
Two side test p value = 0.0004
Cohen's d = 8.7846

----------------------------------------------------

t-test stats for knn vs log:

t = 211.3913
Two side test p value = 0.0
Cohen's d = 172.6003

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