1

COGS 118A: Classification Model Comparison

Linfeng Hu LHU@UCSD.EDU
Department of Cognitive Science
University of California, San Diego
La Jolla, CA 92093, USA

Abstract

This paper describes a replication of part of the the analysis done in the paper by Caruana and Niculescu-Mizil (hereafter referred to as CNM06),(Caruana and Niculescu-Mizil (2006)). This report compares the performances of four different machine learning algorithms on binary classification across four different data sets. The algorithms include: Decision Tree, k-Nearest Neighbors, Logistic Regression and Random Forest. Their performances are measured by three metrics: Accuracy, F1 score and ROC-AUC. .

Keywords: Binary Classification, Decision Tree, k-Nearest Neighbors, Logistic Regression, Random Forest

1. Introduction

As many supervised learning algorithms have been studied and put into use today, it is essential to have a clearer understanding on each of their advantages and disadvantages in dealing with different problems. This report compares and contrasts the following algorithms: Decision Tree, k-Nearest Neighbors, Logistic Regression and Random Forest,(Pedregosa et al. (2011)). To tackle each algorithms' appropriate usage on different types of problems, this report retrieves four different data sets from the UCI Machine Learning repository,(Dua and Graff (2017)).

In addition, this report will also use three different performance metrics(Accuracy, F1, AUC) to assess these supervised learning algorithms. As each different metrics may offer varying results, this report aims to evaluate each algorithm's performance on multiple metrics to measure different trade-offs. This report will also attempt to explore the properties of each metrics and how much each of them will be affected by specific type of training set / testing set differences.

This report's purpose is to present a comparatively smaller-scale replication of the CNM06 paper(Caruana and Niculescu-Mizil (2006)). Caruana and Niculescu-Mizil analyzed the effectiveness of calibration with Platt's Method or Isotonic Regression, which this report did not include and does not concern. They were able to compare nine different algorithms' performance on eleven set problems assessed by eight metrics. From the broader level empirical study done in CNM06, they were able to conclude that without calibration, random forest gives the best performance averaged across all metrics and data sets. Decision trees and logistic regressions has the worst overall performance. KNN shows mediocre overall performance and never exhibit excellent performance on any data sets.

2. Method

2.1 Learning Algorithms

This report attempts to compare the performance of the four chosen learning algorithms. This subsection summarized the different settings I used for each different model in order to find the optimal hyper-parameters.

Decision Tree: I compare two different splitting criterion – the "gini" and "entropy" impurity measures. I also experimented on 5 different max-depth values for the trees – 1, 2, 3, 4, 5.

Logistic Regression: I trained both regularized and not regularized models. The penalization term is 'l2', 'none', with varying the ridge (regularization) parameter by factors of 10 from 10^{-8} *to* 10^{4}.

k-Nearest Neighbors: I used 26 values of K ranging from K = 1 to K = $|trainset|$. I used kNN with Euclidean distance and Euclidean distance weighted by gain ratio. Weights searched were 'uniform', 'distance'.

Random Forest: The forests have 1024 trees. The size of the feature set considered at each split is 1,2,4,6,8,12,16.

These four algorithms are chosen as each of them were specifically mentioned in the conclusion of CNM06 paper, with significantly different performance overall. Thus this report should help replicate the rough results about best, mediocre and worst models as argued in the CNM06 paper.

2.2 Performance Metrics

This report use accuracy, F1 and AUC as performance metrics. To try and have the most statistical power possible when figuring out the differences between algorithms, this report used a paired t-test. I first assign unique random seed to each trial 1-5. And for each algorithm tested, all trial 1s are the same seed, all trial 2s are the same see, etc. Then the test is paired across trials. For each algorithm/data set combination, this report used 5-fold cross-validation via a systematic grid search to find the optimal hyper-parameters setting for each algorithm. Then the trained model is tested on the entire data set excluding the 5000 training data samples. Performance at each stage is measured by accuracy scores.

Accuracy score is the number of correct predictions (true positives and true negatives) divided by the entire population. Accuracy has a fixed threshold at 0.5. This score can range from 0 to 1.

F1 score is calculated as a harmonic mean of both precision score and recall score. Thus it seeks a balance and inclusion of both. It is often more useful than accuracy when the data set is uneven (little positives and overwhelmingly many negatives). This score can range from 0 to 1.

AUC(Area Under the Curve) score measures the area under the ROC curve. It can range from 0 to 1. A ROC curve plots true positive rate against false positive rate. Thus, AUC score measures how well predictions are ranked, thus is scale-invariant. (Courses (2020))

In general, the higher these scores are, the better the performance it foretells.

2.3 Data Sets

This report compares the algorithms on four binary classification problems. Four data sets are all retrieved from the UCI Machine Learning repository: ADULT, BANK, LETTER, and COVTYPE (see Table 1 for more information).

Table 1: Description of Problems

NAME	#ATTR	TRAIN SIZE	TEST SIZE	%POZ
LETTER	16	5000	20000	0.503
ADULT	14(114)	5000	32561	0.759
COVTYPE	12(54)	5000	581012	0.035
BANK	16(60)	5000	45211	0.117

LETTER: I used letter A-M as negatives and the rest as positives to convert this data set into a balanced, binary classification problem.

ADULT: This is a binary classification problem as retrieved. It is a slightly unbalanced data set with 75.9% positive data. Attribute "Sex" is binary thus use label encoding. Other categorical attributes use one-hot encoding to transform into numerical data.

COVTYPE: This is originally a multi-class data set as retrieved. I processed the data as Caruana and Niculescu-Mizil did. This dataset is converted into a binary problem by treating the largest class – 7 – as positive, and the rest as negative. Thus we have a severely unbalanced data set with merely 3.5% of the data as positive.

BANK: This is a binary problem as retrieved. Categorical data is converted using label and one-hot encoding. The original data set is highly unbalanced with only 0.117% of data labeled as positive.

3. Experiment & Discussion of Results

3.1 Primary Results

As this report has briefly stated before, for each algorithm/data set combination, I randomly choose 5000 data samples for my training set with replacement. I then do 5-fold cross-validation on this selected training set to find out the hyper-parameters via a systematic grid-search of the parameter space best for this particular algorithm. In the section above I have listed the hyper-parameter values I used in my search for each of the four algorithms.

Main results from this experiment is presented in the following Tables, each of them will be discussed in the following paragraphs. In these tables, the algorithm with the best performance on each metric is boldfaced. Other algorithm's whose performance is not statistically distinguishable from the best algorithm at p = 0.05 using paired t-tests on the five trials are *'ed. The process of paired t-test can be found the section 2.2. Like the CNM06 paper states, the entries in these tables that are neither boldfaced nor starred indicate performance that is significantly lower than the best models at p = 0.05.

Table 2: Performance by Metric

NAME	ACC	AUC	F1	MEAN
DT	0.867	0.716	0.653	0.745
kNN	0.904	0.711	0.644	0.753
LR	0.844	0.617	0.502	0.654
RF	**0.929**	**0.809**	**0.770**	**0.836**

As Table 2 shows, random forest gives the best mean performance over all problem sets and across all three metrics, which results in a best MEAN performance. All other results are not starred, which means that none of these other algorithms' performance surpassed the threshold of p = 0.05. The mean test set performances across trials for decision trees, kNN and logistic regression are all significantly lower than the best model (RF) here at p=0.05, when compared using paired t-test.

The results for the three algorithms other than random forest also matches the results from CNM06 which stated that logistic regression and decision trees are the worst performing models – they give the lowest MEAN test set performances across trials here. While memory-based methods such as kNN is mediocre – it gives the second-best mean performance in this report, but still far worse than random forest. The raw test set performance can be found in Appendix 2. The exact p-values as results of paired t-tests here can be found in Appendix 3.

Table 3: Performance by Problem

	ADULT	BANK	COVTYPE	LETTER	MEAN
DT	0.825	0.666	0.728	0.761	0.745
KNN	0.751	0.590	0.706	**0.963**	0.753
LR	0.764	0.591	0.537	0.727	0.655
RF	**0.863**	**0.714**	**0.814**	0.953*	**0.836**

As Table 3 shows, random forest still gives the best MEAN test set performance across these four data sets as it is the best-performing algorithm for data set. ADULT, BANK and COVTYPE. However, kNN outperforms random forest for the LETTER data set, although there is no statistically significant distinguishable difference for the LETTER data set between kNN and random forest. The exact P-Values from paired t-test can be seen in Appendix 4. As I described in Table 1 before, LETTER data set is the only roughly balanced binary data set among our choices (with 50.3% positive data). This trait may allow for easier, more likely successful training of the data. LETTER is also the smallest data set used here which could also help explain why it yields statistically indistinguishable performance from two algorithms here. Initially, I imagined that kNN would perform well on the COVTYPE data set among all since that is the most imbalanced data set with only 3.5% positive label for algorithms to train on and kNN's algorithm should not be affected by the size of the positive class here. But kNN did not perform as well as in principal, it may be due to the fact that COVTYPE is also the largest data set among all four, more than ten times larger than the remaining three data sets. This fact may result in undesirable initialization and further affect the ultimate outcomes.

Also, one can observe that in general, all four algorithms performed undesirably on the data sets BANK and COVTYPE. This is due to the fact that these two are the two most

imbalanced problems, each with less than 15% positive labeled data. One can see that logistic regression is especially strongly affected by such imbalance. This may be due to the fact that logistic regression is considered a generalized linear model and its learning outcome is likely influenced by unable to appropriately learn the positive class during training.

3.2 Secondary Results

As Appendix 1 at the end shows, algorithms' train set performance and test set performance has a lot similarities. It is the same that for the accuracy, AUC and F1 scores, random forest still consistently achieved the highest scores across all four data sets with optimal hyper-parameters. It is also the same that F1 scores are almost always the lowest among all algorithm-data set combinations as compared to the other two metrics. The F1 scores are especially bad for BANK and COVTYPE data sets because they are extremely imbalanced. But the "ranking" of the algorithms within each data set is not altered. The following is a table(Table 5) that calculated the mean training performance across data sets.

Table 4: Mean Training Performance across data sets with Optimal Hyper-parameters

	ACC	AUC	F1
DT	0.847131	0.82142118034	0.57414255919
KNN	0.83735980769	0.73829120479	0.42813081044
LR	0.83011142857	0.7947041979	0.47693179496
RF	**0.92063142857**	**0.94679692101**	**0.71748636913**

One can further see from this table 4 that decision tree and random forest gives their best performances when measured by AUC metric, kNN and logistic when measured by accuracy. It can be seen here that there is a larger discrepancy in the F1 score between random forest and the other three algorithms. Also, from Appendix 1 with a more comprehensive view on the training data, we can see that when using the metric accuracy, algorithms' performances does not seem to be have a huge variance. While measured using AUC or F1 scores, there may be discrepancies as large as 0.3 to 0.4. This discrepancy can be observed between random forest's and kNN's F1 scores on COVTYPE and BANK data sets. This may be the reason why random forest is able to score much higher (significantly distinguishable) on F1 score.

Table 5: Very "Rough" Time Used for each algo/data combo

	ADULT	BANK	COVTYPE	LETTER
DT	00:02	00:02	00:07	00:01
KNN	**08:07**	**04:25**	**19:49**	**03:19**
LR	00:10	00:08	00:16	00:04
RF	00:33	00:33	01:13	00:35

As Table 5 shows, the time complexity for different algorithm-data set combinations can vary a lot. The training time for decision tree has been the shortest across all problems. The training time for COVTYPE data set is the highest among all data set. This may be attribute to its extremely imbalanced nature and learning algorithms' difficulty in learning positive class as a result.

The training time for kNN is consistently the highest across all data sets. This may be attributed to its bias-variance trade-off mentioned in class. Since kNN is a memory-based methods that does not have assumption on the model to use on the given data distribution (as mentioned in CNM06), it has low bias. However, that means that kNN has a high variance, which would result in kNN paying too much attention to the training data (including the noise). This makes kNN computationally expensive as it stores all the memory of the training data.

In addition, we may also generalize some knowledge from heatmap-style plots of the validation performances vs hyper-parameter settings for these 4 algorithms (plots can be seen in Appendix B). The F1 scores from data set BANK and COVTYPE across all algorithms and their hyper-parameter settings are awful because of the small positive class they have. F1 score ignores the value of true negatives in outputs. All algorithms do not have largely awful performance on the LETTER data set. This may be attribute to the well-balanced data distribution. Decision tree generally performed well on the two highly imbalanced data set. The best performing hyper-parameter settings seem to be {'max_depth' = 5} regardless of the 'criterion' parameter. kNN having the setting {'weights' = 'distance'}, meaning weight points by the inverse of their distance. "In this case, closer neighbors of a query point will have a greater influence than neighbors which are further away",(Pedregosa et al. (2011)). This setting consistently outperforms kNN with uniform weights for all points in each neighborhood. Logistic regression has undesirable performance when C value is set lower, especially when C value is in the range of 1e-05 to 1e-08. Random forest performs very well in general as we can see that the whole range of scores is higher than other heatmaps. The worst hyper-parameter setting would be when 'max_features' is set to 1, meaning the size of the random subsets of features to consider when splitting a node is very little. In which case it yields the worst performance across all four data sets.

4. Conclusion

In conclusion, this report attempts to replicate the CNM06 paper and further validate its results prior to calibration. This report's findings are in general similar to Caruana and Niculescu-Mizil's findings. Random forest does give the best overall performance and almost over-performed all other algorithms on each problem, across all three metrics. And, on average, memory-based learning (kNN), decision tress and logistic regression are not competitive with the best method as evidenced by high p-values in paired t-test. The only statistically indistinguishable performance given is by kNN and random forest on the LETTER data set as they all achieved high performance scores. On average, logistic regression gives the worst mean test set performance across trials for each algorithm/metric combo. And kNN gives the more variable performance depending on different data set (different data distributions). Although kNN has the lowest performance across all 4 algorithms for BANK data set, it performed the best on the LETTER data set which ultimately allows its mean performance to overpower logistic regression.

Aside from the comparison among algorithms, one may also observe that due the highly imbalanced nature of BANK and COVTYPE data set (both with small percentage of positive data), F1 score always gives the worst mean performance by metrics. This is due to the fact that these 2 imbalanced data sets dragging the overall F1 scores down. LETTER

data set, as the most balanced one with 50.3% positive data, allows all four algorithms to perform their bests.

Therefore, despite this report is restricted to less algorithms, less data sets and less metrics used, it still upholds the CNM06 paper's finding in general. The code used is attached at the very end of this report.

5. Bonus

In addition to the 3 required algorithms, this report explored the use of a fourth algorithm.

This report also includes an extra time complexity analysis on these 4 algorithms.

You can find the heatmap-style plots of the validation performance vs hyper-parameter setting for these 4 algorithms in the Appendix B.

Acknowledgments

I would like to acknowledge the support for this project from the COGS 118A Instructional Team and the anonymous help provided by the two students whose papers are used as sample reports for our guidance. In addition, I would also like to thank the support and help I received from my fellow classmates on Piazza and Discord over this entire quarter.

Appendix A

In this appendix I provide the tables from secondary results:

Appendix 1: Training Performance with Optimal Hyper-parameters

Dataset	Algorithm	ACC	AUC	F1
ADULT	DT	0.8175760000000000	0.837423678512884	0.8877002541069490
ADULT	KNN	0.7707192307692320	0.6023583482806860	0.867350448452771
ADULT	LR	0.7906428571428570	0.6754622783820800	0.8738581381097860
ADULT	RF	0.8581085714285710	0.9091913036591850	0.9084760972324240
BANK	DT	0.8937040000000000	0.78130124826662	0.3475866369243280
BANK	KNN	0.887494615384616	0.7569990603502410	0.09075517219374930
BANK	LR	0.8882000000000000	0.8290119827249890	0.2474943716992440
BANK	RF	0.9061771428571430	0.9125190386343290	0.4363711622152500
COVTYPE	DT	0.9687000000000000	0.8943617959139150	0.37062970216110800
COVTYPE	KNN	0.9654276923076950	0.7791546270469900	0.024560746094349200
COVTYPE	LR	0.9650114285714290	0.8829566051995300	0.12527881046351900
COVTYPE	RF	0.978525714285714	0.9769847004736370	0.5856145972399730
LETTER	DT	0.708544	0.7725979986885840	0.6906536435678810
LETTER	KNN	0.7257976923076930	0.8146527835107710	0.7298568750376180
LETTER	LR	0.6765914285714290	0.791385925324797	0.6610958595724490
LETTER	RF	0.9397142857142860	0.9884926413016210	0.9394836198519930

Appendix 2: Raw Test Set Score

Dataset	Algorithm	ACC	AUC	F1
ADULT	DT	0.842586734641647	0.7307772297360420	0.9012824874553120
ADULT	KNN	0.7937061720053640	0.5811695428711440	0.8794516281183990
ADULT	LR	0.7969738849134450	0.6155360621368030	0.8783268316122800
ADULT	RF	0.8701104593429770	0.8024420685077620	0.9160105816305150
BANK	DT	0.8976591242544220	0.6599912349722280	0.4410955776184040
BANK	KNN	0.8857387214026080	0.5960083792878690	0.28893520307165400
BANK	LR	0.8879726910117740	0.589651752599677	0.2940201467824260
BANK	RF	0.911295923558426	0.7014415167621860	0.5292165617376660
COVTTYPE	DT	0.9720460162612820	0.7157839924396490	0.4950844768726670
COVTTYPE	KNN	0.973013523530208	0.7039614060577880	0.44213062910249300
COVTTYPE	LR	0.9640421655089170	0.5344377984785100	0.111494523849969
COVTTYPE	RF	0.9817262523550860	0.7776771976694800	0.6819420670936040
LETTER	DT	0.7559233333333330	0.7554984712782990	0.7726514982630080
LETTER	KNN	0.9634200000000000	0.963419883115792	0.963629906745131
LETTER	LR	0.72839	0.7285092063314280	0.7241041362843800
LETTER	RF	0.9526966666666670	0.952705664070573	0.9528936005480940

Appendix 3: P-Values for Table 2

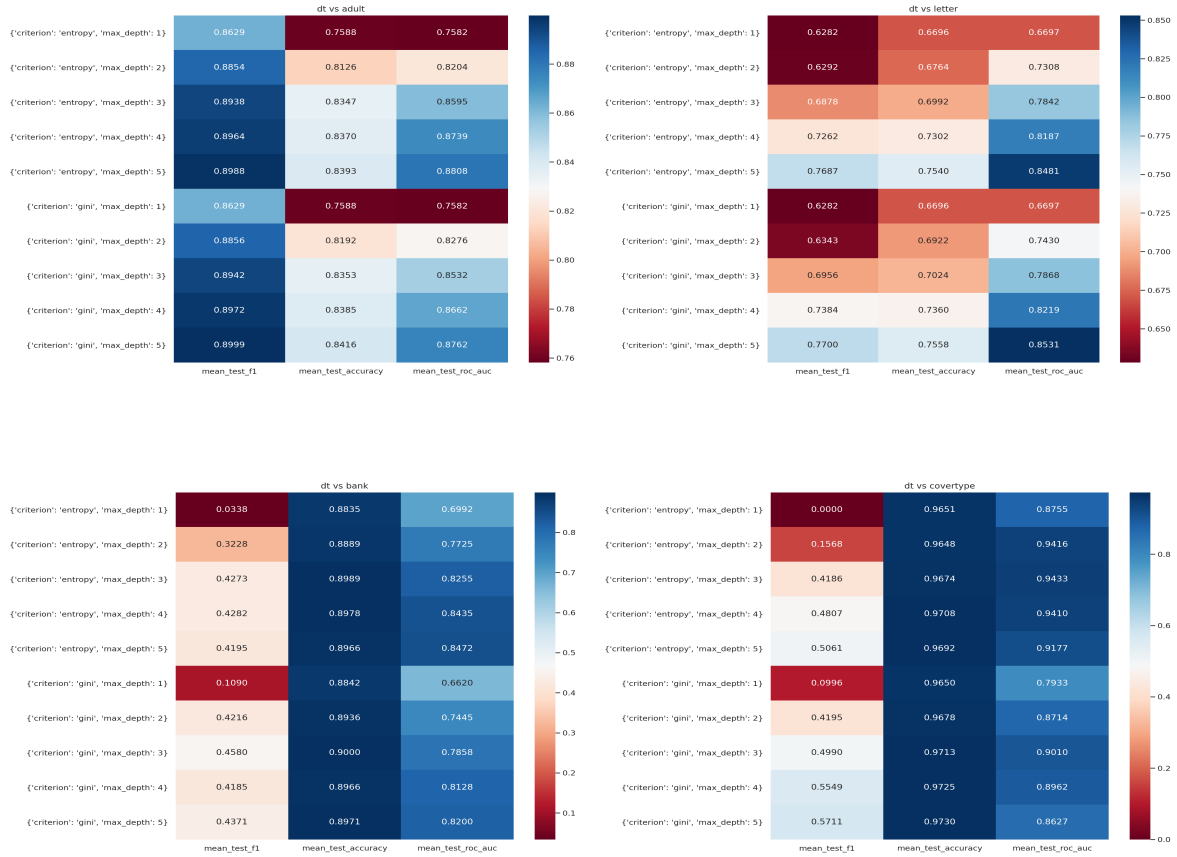
Algorithm	ACC	F1	AUC
RF	Inapplicable	Inapplicable	Inapplicable
LR	0.0003581013150458760	0.00032220171891499500	0.00023156740370248600
DT	0.0036367644837718300	0.0034176864471131700	0.0019262760372985500
kNN	0.012363506697798300	0.0014316380060976900	0.0034075707747724200

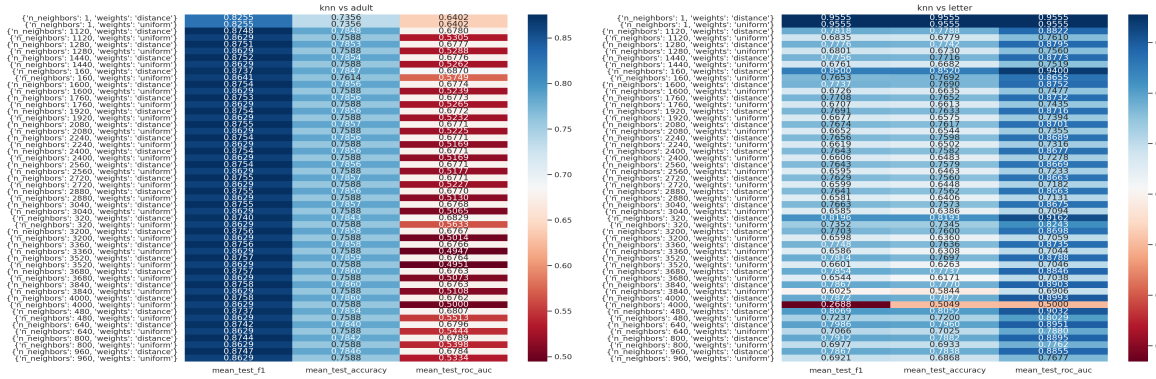
Appendix 4: P-Values for Table 3

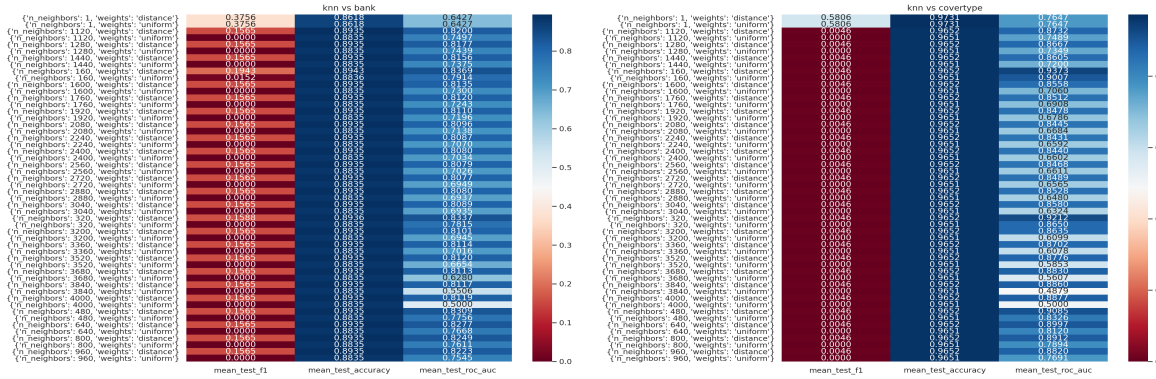
	ADULT	BANK	COVTYPE	LETTER
RF	Inapplicable	Inapplicable	Inapplicable	0.091784466
LR	2.69e(-6)	6.51e(-6)	2.43e(-6)	0.003218953
DT	0.00034037	1.09e(-4)	0.000593927	0.004577588
kNN	2.08e(-7)	5.38e(-6)	0.004577588	Inapplicable

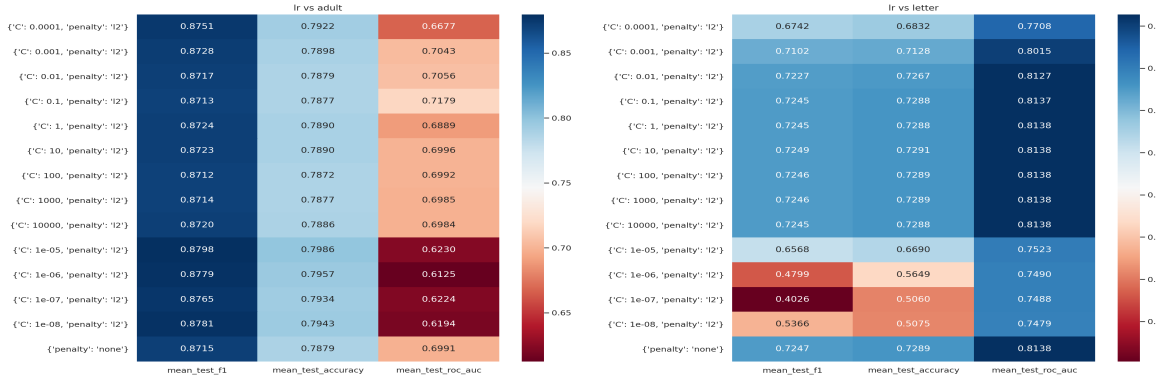
Appendix B

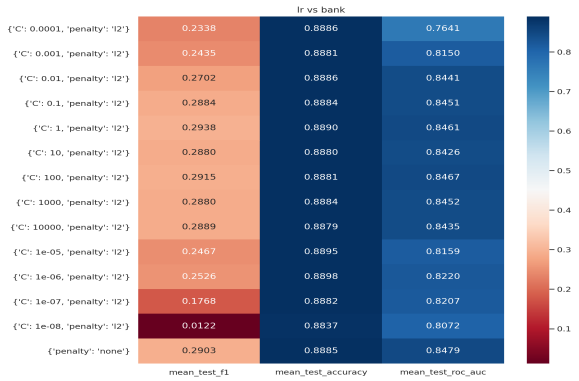
In this appendix I provide Extra Credit heatmap-style plot of the validation performance vs hyper-parameter setting for my algorithms:

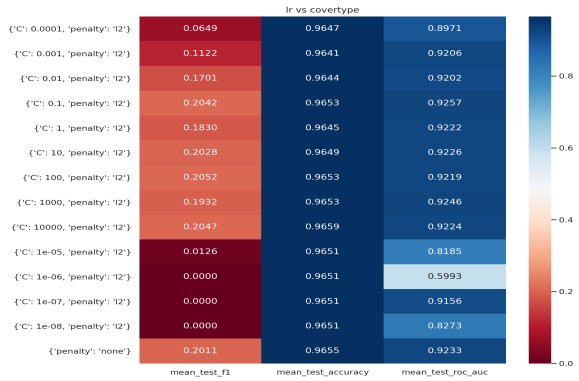


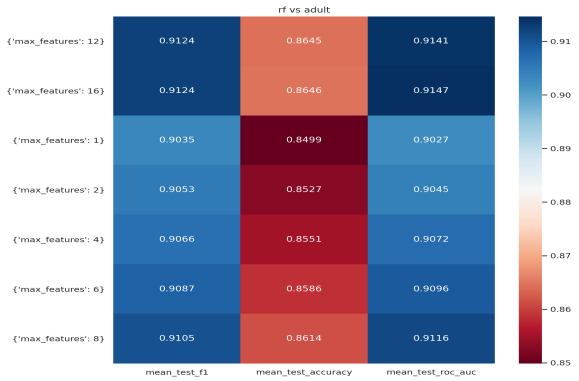

knn vs adult
knn vs letter
mean_test_f1
mean_test_accuracy
mean_test_roc_auc

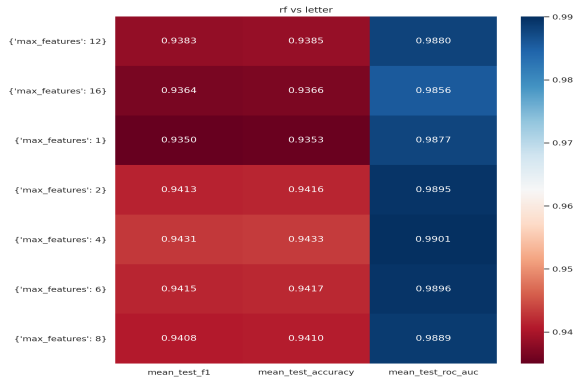

knn vs bank
knn vs covertype
mean_test_f1
mean_test_accuracy
mean_test_roc_auc

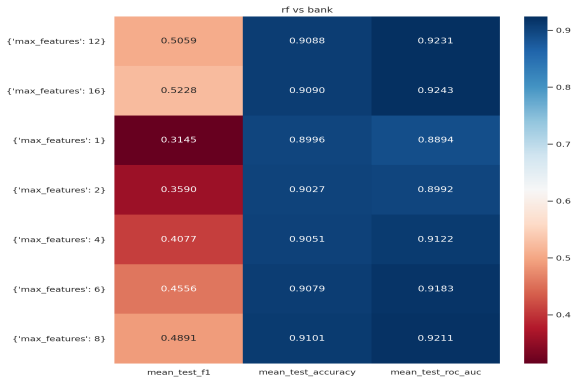

lr vs adult
lr vs letter
mean_test_f1
mean_test_accuracy
mean_test_roc_auc

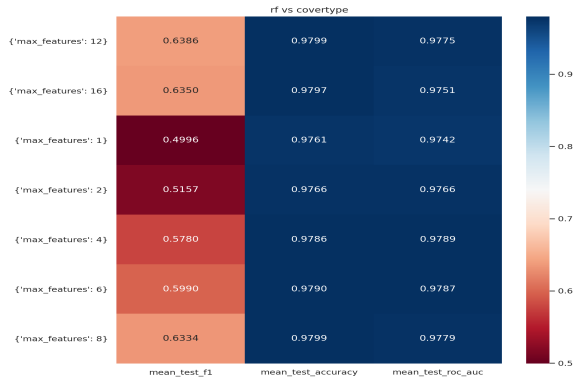












CODE is attached after the references

References

R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. *In Proceedings of the 23rd international conference on Machine learning*, 161-168, 2006.

Machine Learning Crash Courses. Classification: Roc curve and auc, 2020. URL `https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc`.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

main

March 17, 2021

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

from sklearn.utils import resample
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
import json
from os import sep
import pickle as plk
from pathlib import Path
from tqdm import tqdm

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

[2]:
```
root = Path('dataset')
save = Path('save')
```

[5]:
```
score = ['accuracy', 'roc_auc', 'f1']
```

[4]:
```
logs = ""
best_res = []
raw_res = []
train_res = []
```

[5]:
```
def getLetterData(root):
    df = pd.read_csv(root/'letter'/'letter-recognition.data', header=None)
    raw_y = df[0]
    raw_X = df[[x for x in df.columns if x != 0]]
    raw_y = raw_y.apply(lambda x: 0 if x < 'N' else 1)
```

```
    return raw_y,raw_X
def getCovData(root):
    data = pd.read_csv(root/'cov'/'covtype.data', header=None)
    raw_y = data[54].apply(lambda x: 0 if x!=7 else 1)
    raw_X = data[[x for x in data.columns if x != 54]]
    return raw_y, raw_X
def getAdultData(root):
    data = pd.read_csv(root/'adult'/'adult.data', header=None)
    raw_y = data[14].apply(lambda x: 0 if x!=' <=50K' else 1)
    exp_col  = 9 # sex
    num_cols = [0, 2, 4, 10, 11, 12]
    cat_cols = [x for x in data.columns if x not in (num_cols + [14, exp_col])]

    onehotEnc = OneHotEncoder()
    onehotArr = onehotEnc.fit_transform(data[cat_cols])
    for col in (cat_cols + [exp_col]):
        data[col] = LabelEncoder().fit_transform(data[col])
    raw_X = data[[x for x in data.columns if x != 14]]
    raw_X = np.concatenate((raw_X, onehotArr.toarray()), axis=1)
    return raw_y, raw_X
def getBankData(root):
    data = pd.read_csv(root/'bank'/'bank-full.csv', sep=';')
    target = 'y'
    num_cols = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays',␣
 ↪'previous']
    cat_cols = [x for x in data.columns if x not in (num_cols+[target])]
    onehotEnc = OneHotEncoder()
    onehotArr = onehotEnc.fit_transform(data[cat_cols])
    for col in cat_cols:
        data[col] = LabelEncoder().fit_transform(data[col])
    raw_X = data[[x for x in data.columns if x != target]]
    raw_X = np.concatenate((raw_X, onehotArr.toarray()), axis=1)
    raw_y = data[target].apply(lambda x: 0 if x=='no' else 1)
    return raw_y, raw_X
```

[6]:
```
def calMetric(raw_y, y_p):
    acc = accuracy_score(raw_y, y_p)
    auc = roc_auc_score(raw_y, y_p)
    f1 = f1_score(raw_y, y_p)
    return acc, auc, f1

def oneTrial(X, y, raw_X, raw_y, model, params, model_name, idx, dataname):
    global best_res, raw_res

    clf = model()

    cv = GridSearchCV(clf, params, cv=5, n_jobs=-1,
```

```
                        scoring=score, refit=False)
    cv.fit(X, y)
    result = cv.cv_results_

    name = f"{dataname}-{model_name}-run {idx}"
    res = saveResult(result, name)
    res['num_run'] = idx
    res['model_name'] = model_name
    res['data_name'] = dataname
    raw_res.append(res)

    for sc in score:
        best_idx = result['rank_test_'+sc].argmin()
        params = result['params'][best_idx]
        clf = model(**params)
        clf.fit(X, y)
        y_p = clf.predict(raw_X)
        acc, auc, f1 = calMetric(raw_y, y_p)
        best_res.append([idx, model_name, sc, acc, auc, f1, dataname])
        y_p = clf.predict(X)
        acc, auc, f1 = calMetric(y, y_p)
        train_res.append([idx, model_name, sc, acc, auc, f1, dataname])
```

[7]:
```
def saveResult(result, name):
    keys = list(result.keys())
    for idx in keys:
        if idx.startswith('param_'):
            result.pop(idx)

    result = pd.DataFrame(result)
    return result
```

[8]:
```
n_neighbors = np.floor(np.linspace(1, 4000, 26)).astype(int)

algorithm_pair = [
    (
        RandomForestClassifier,
        {'max_features':[1, 2, 4, 6, 8, 12, 16]},
        'rf'
    ),
    (
        LogisticRegression,
        [{'penalty': ['none']}, {'C': [10**x for x in range(-8, 5)], 'penalty':␣
 ↪['l2']}],
        'lr'
    ),
    (
```

```
        DecisionTreeClassifier,
        {'criterion': ("gini", "entropy"), 'max_depth': [1, 2, 3, 4, 5]},
        'dt'
    ),
    (
        KNeighborsClassifier,
        {'n_neighbors':n_neighbors.tolist(), 'weights': ('uniform', 'distance')␣
 ↪},
        'knn'
    )
]
```

[ ]:
```
datalist = [(getLetterData(root),   'letter'),
            (getCovData(root),      'covertype'),
            (getAdultData(root),    'adult'),
            (getBankData(root),     'bank')]
for (raw_y, raw_X), dataname in datalist:
    for algorithm, params, name in algorithm_pair:
        for i in tqdm(range(5), desc=dataname + '-' + name):
            X, y = resample(raw_X, raw_y, n_samples=5000, random_state=i)
            oneTrial(X, y, raw_X, raw_y, algorithm, params, name, i+1, dataname)
```

[ ]:
```
best_res = pd.DataFrame(best_res, columns=['num_run', 'model', 'select metric',␣
 ↪'accuracy', 'auc', 'f1', 'dataset'])
raw_res = pd.concat(raw_res)
```

[ ]:
```
best_res.to_csv(save / 'best.csv')
raw_res.to_csv(save / 'raw.csv')
```

[ ]:
```
table1 = best_res.groupby(['dataset', 'model']).mean().reset_index()
table2 = best_res.groupby(['dataset', 'model', 'select metric']).mean().
 ↪reset_index()
table2_1 = best_res.groupby(['model', 'select metric']).mean().reset_index()
table1[['dataset', 'model', 'accuracy', 'auc', 'f1']].to_csv(save / 'table1.
 ↪csv', sep=',', index=False)
table2[['dataset', 'model', 'select metric', 'accuracy', 'auc', 'f1']].
 ↪to_csv(save / 'table2.csv', sep=',', index=False)
table2_1[['model', 'select metric', 'accuracy', 'auc', 'f1']].to_csv(save /␣
 ↪'table2-1.csv', sep=',', index=False)

table3 = raw_res.groupby(['data_name', 'model_name']).mean().reset_index()
table3 = table3[['data_name', 'model_name', 'mean_test_accuracy',␣
 ↪'mean_test_roc_auc', 'mean_test_f1']]
table3.columns = ['dataset', 'model', 'accuracy', 'auc', 'f1']
table3.to_csv(save / 'table3.csv', sep=',', index=False)
```

[ ]:
```
from scipy.stats import ttest_rel
```

[ ]:
```
best = pd.read_csv(save / 'best.csv')
```

[ ]:
```
algo = ['rf', 'lr', 'dt', 'knn']

p_value = pd.DataFrame(algo, columns=['algo'])
t_value = pd.DataFrame(algo, columns=['algo'])

best_aglo = 'rf'

def get_t_p(best, best_aglo='rf'):
    p_value = pd.DataFrame(algo, columns=['algo'])
    t_value = pd.DataFrame(algo, columns=['algo'])

    for metrics in ['accuracy', 'f1', 'roc_auc']:
        sample = best[(best['2'] == metrics)]
        get_algo_res = lambda algo, sample: sample[sample['1'] == algo]['3']
        t_value[metrics] = [ttest_rel(get_algo_res(best_aglo, sample),␣
 ↪get_algo_res(a, sample))[0] for a in algo]
        p_value[metrics] = [ttest_rel(get_algo_res(best_aglo, sample),␣
 ↪get_algo_res(a, sample))[1]    for a in algo]
    p_value = p_value.fillna(0)
    t_value = p_value.fillna(0)
    return p_value, t_value

p_value, t_value = get_t_p(best)
p_value.to_csv(save / 'p_val.csv', index=False)
t_value.to_csv(save / 't_val.csv', index=False)
```

[ ]:
```
p_value = pd.DataFrame()
t_value = pd.DataFrame()

for i in ['adult', 'bank', 'covertype']:
    p_v, t_v = get_t_p(best[best['6'] == i])
    p_v['dataset'] = i
    t_v['dataset'] = i
    p_value, t_value = pd.concat((p_value, p_v)), pd.concat((t_value, t_v))

p_v, t_v = get_t_p(best[best['6'] == i], 'knn')
p_v['dataset'] = i
t_v['dataset'] = i
p_value, t_value = pd.concat((p_value, p_v)), pd.concat((t_value, t_v))
p_value.to_csv(save / 'p_val_full.csv', index=False)
t_value.to_csv(save / 't_val_full.csv', index=False)
```

[7]:
```
raw_res = pd.read_csv(save / 'raw.csv')
raw_res.head()
```

[7]:

	Unnamed: 0	mean_fit_time	std_fit_time	mean_score_time	std_score_time	\
0	0	0.371324	0.115417	0.059528	0.014269	
1	1	0.441922	0.081536	0.059999	0.012082	
2	2	0.434682	0.079569	0.045988	0.008635	
3	3	0.566187	0.096903	0.051658	0.007580	
4	4	0.671549	0.093789	0.039611	0.000983	

	params	split0_test_accuracy	split1_test_accuracy	\
0	{'max_features': 1}	0.924	0.935	
1	{'max_features': 2}	0.928	0.942	
2	{'max_features': 4}	0.925	0.946	
3	{'max_features': 6}	0.925	0.952	
4	{'max_features': 8}	0.920	0.947	

	split2_test_accuracy	split3_test_accuracy	…	split1_test_f1	\
0	0.936	0.939	…	0.934410	
1	0.944	0.938	…	0.941296	
2	0.940	0.944	…	0.946000	
3	0.944	0.941	…	0.952191	
4	0.946	0.942	…	0.947159	

	split2_test_f1	split3_test_f1	split4_test_f1	mean_test_f1	std_test_f1	\
0	0.935484	0.938939	0.932406	0.932610	0.005800	
1	0.943548	0.937876	0.938614	0.937482	0.006047	
2	0.939516	0.943888	0.940476	0.938591	0.008100	
3	0.943548	0.940941	0.943396	0.940787	0.009286	
4	0.945455	0.941650	0.940594	0.938578	0.010550	

	rank_test_f1	num_run	model_name	data_name
0	7	1	rf	letter
1	5	1	rf	letter
2	2	1	rf	letter
3	1	1	rf	letter
4	3	1	rf	letter

[5 rows x 33 columns]

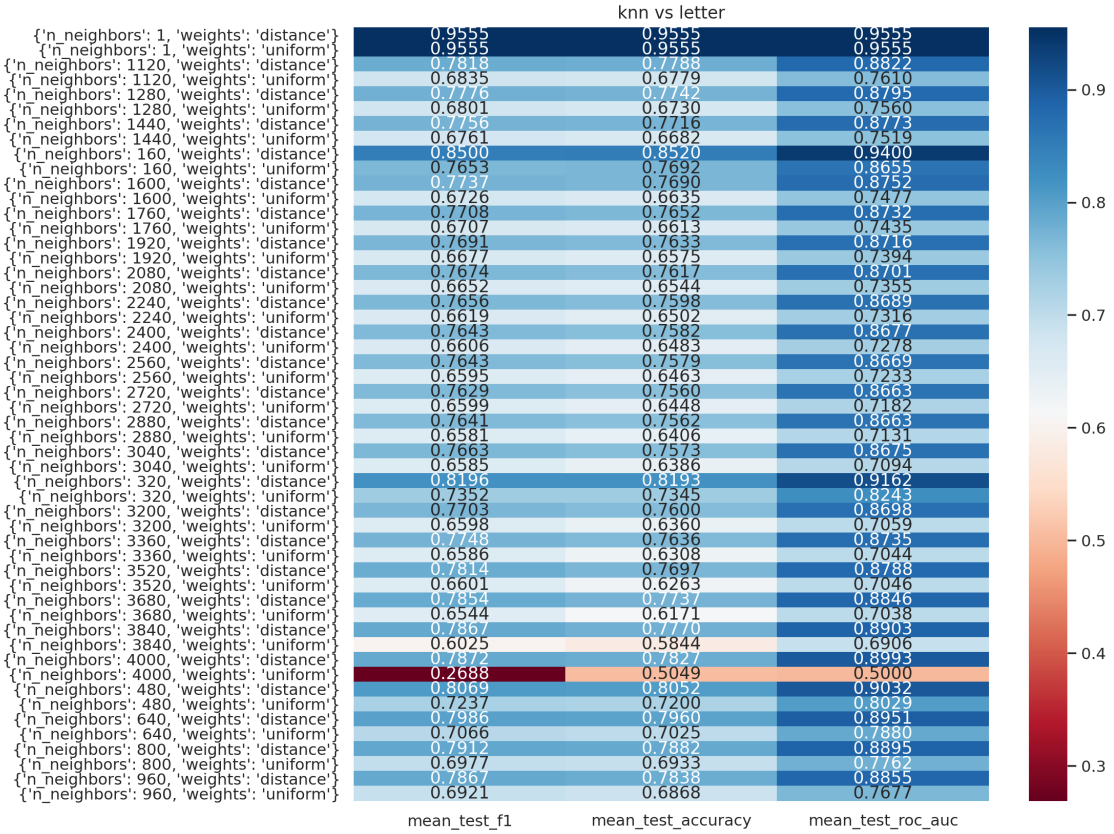
[8]:
```
import seaborn as sns
sns.set()
import matplotlib.pyplot as plt
```

[9]:
```
res = raw_res.groupby(['model_name', 'data_name', 'params']).mean().
↪reset_index()
```

[19]:
```
def get_heatmap(m, d):
    one = res[(res['model_name'] == m) & (res['data_name'] == d)]
    f, ax = plt.subplots(figsize=(10,10), dpi=200)
    sns.heatmap(one[['mean_test_f1', 'mean_test_accuracy',␣
 ↪'mean_test_roc_auc']], fmt="0.4f", cmap = 'RdBu', ax=ax,␣
 ↪yticklabels=one['params'], annot=True)
    name = f"{m} vs {d}"

    label_y = ax.get_yticklabels()
    plt.setp(label_y, rotation=0, horizontalalignment='right')
    # ax.set_ylim([7, 0])
    plt.title(name)
    plt.savefig(save / 'img' / (name + '.png'), dpi=200, bbox_inches = 'tight')
```

[20]:
```
for m in ['knn', 'lr', 'rf', 'dt']:
    for d in ['letter', 'covertype', 'bank', 'adult']:
        get_heatmap(m, d)
```



& many pages of heatmaps like this