COGS118A: An effort to replicate CMN06

Nguyen Hoang NMHOANG@UCSD.EDU
Department of Mathematics
University of California, San Diego
San Diego, CA 92093, USA

Editor: Nguyen Hoang

Abstract

Taking Caruana and Niculescu-Mizil (2006) as a reference, this paper is my attempt in order to replicate a portion of its methods and results produced, using 4 different learning algorithms and 5 different data sets taken from UCI Machine Learning Repository over 3 different metrics. The algorithms consist of k-nearest neighbors, neural network, random forest and logistic regression.

Keywords: Logistic Regression, K-Nearest Neighbors, Random Forest, neural network

1. Introduction

Since Caruana and Niculescu-Mizil (2006), there has been a lot more comprehensive empirical studies comparing learning algorithms. However, replicability still plays an important role even for old papers. Hence, this project is my effort in order to present an analysis of old methods using new computational modules like Scikit-learn(Pedregosa et al. (2011)). The results of this project is mostly consistent with Caruana and Niculescu-Mizil (2006) with some slight negligible difference. Table 2 shows the performances over problems for each algorithm, where the Random Forest outperform on most dataset, with some insignificant difference with ANN if it was outperformed, followed by KNN and the worst of all is LOGREG. While in table 2, the random forest is the sole winner over 3 metrics, followed by ANN and KNN with some small but considered significant difference.

2. Methods

2.1 Learning algorithms

For the algorithms that I used, KNN from memory-based method, ANN from supervised neural networks, Random Forest from ensemble method and Logistic Regression from linear model method. I chose the hyper-parameters closely followed with those from CNM06 with minor changes.

KNN: I used 26 values of K ranging from 1 to 105 equally spaced. I only used KNN with Euclidean distance, with both uniform weighted and distance weighted.

ANN: I train neural nets with both stochastic gradient descent and adam backpropagation. The number of hidden units is varied 1,2,4,8,32,128. The number of batch size is 200, RELU is the main activation function and the maximum number of iterations is 500. The momentum of SGD is varied between 0, 0.2, 0.5 and 0.9

Random Forest: The forests have 1024 trees. The size of the feature set considered at each split is 1,2,4,6,8,10 and square root of the number of attributes.
Logistic Regression (LOGREG): In order to encourage convergence in trade-off for speed, I set the maximum iterations to 5000. While training both unregularized and regularized models, I varied the regularization parameter from 10^{-8} to 10^{4}.

2.2 Metrics

The metrics that I used for this paper are AUC-ROC, Accuracy and F1 score. Furthermore, metric Accuracy will be the base measure for comparisons between models and data sets. However, the metric alone might only captures a small picture of the whole, I have also include AUC-ROC and F1 for reference.

2.3 Data Sets

The five data sets used are ADULT, COV, LETTER.P2 like in Caruana and Niculescu-Mizil (2006) and AVILA, BEAN from the UCI Machine Learning Repository. Table 1 list the descriptions of the data sets. Just like in Caruana and Niculescu-Mizil (2006), COV data set from Blackard et al. (1998) has been converted to a binary problem by treating the largest class as the positive and the rest as negative, LETTER.p2 from Slate (1991) uses letters A-M as positives and the rest as negatives, yielding a well balanced problem. ADULT from Kohavi and Becker (1994) contains categorical features, hence I used one-hot-encoding in order to convert them to multiple binary features. AVILA data set from C. DeÂ Stefano (2018) has already been normalized by using the Z-normalization method so no processing is needed. Similar to COV data set, BEAN from Koklu and Ozkan (2020) has also been converted to a binary problem by treating the largest class as the positive and the rest as negative.

3. Experiment

3.1 Set-up

For each data set and algorithm combination, 5000 data samples are randomly sampled from 5 different times to fit into a 5-fold cross validation GridSearchCV in order to find the best hyper-parameters to optimize the corresponding score metrics. After having the best setting for each metrics, I used them to retrain the classifier into three different models, each optimize one metric. Afterwards, the models are used to make prediction on both training set and testing set, yielding both training and testing scores for 3 metrics. After which, the process is repeated for every classifier-dataset pair to get the desirable statistics for the tables below.

3.2 Results

Table 2 and 3 below are used to present the performance. The metrics score of the best-performing algorithm per data set are boldfaced while asterisks are used to denote algorithms that has near-similar performance with the best.

4. Conclusion

The best classifier for both data sets COV and AVILA is Random Forest while for ADULT and BEAN data sets, the ANN triumphs and KNN is the winner for LETTER data set. However, if we look closely, the datasets ADULT, LETTER and BEAN has achived performance that are not much significantly differs, except for LOGREG on LETTER. Over scoring metrics, RF uniformly outperforms the other classifiers, followed with ANN and KNN while LOGREG performing uniformly the worst of all.
If we were to compare Table 2 and Table 4, we can see that the performances of LOGREG on both training and testing set express no difference on all 3 metrics. This shows that LOGREG can be a bad choice to be chosen as a algorithm for binary classification problem compared to the other 3.

Tables

List of Tables

Table 1: Description of datasets

Dataset	#Attr	Train size	Test size	%Poz
ADULT	14/108	5000	25725	24.9%
COV	54	5000	531012	48.76%
LETTER.P2	17	5000	15000	49.7%
AVILA	11	5000	15867	41.08%
BEAN	17	5000	8611	26.05%

Table 2: Mean test set performance across trials over metrics

Model	AUC	Accuracy	F1
KNN	0.843	0.865	0.809
NN	0.861*	0.879*	0.827
RF	**0.890**	**0.907**	**0.862**
LOGREG	0.770	0.794	0.725

Table 3: mean test set performance across trials over by data set

Model	ADULT	COV	LETTER.P2	AVILA	BEAN
KNN	0.737	0.784	**0.952**	0.785	0.937*
NN	**0.765**	0.799	0.947*	0.825	**0.943**
RF	0.761*	**0.820**	0.947*	**0.965**	0.940*
LOGREG	0.757*	0.754	0.729	0.634	0.941*

Table 4: Mean training set performance across trials over metrics

Model	AUC	Accuracy	F1
KNN	0.961	0.982	0.960
NN	0.911	0.928	0.879
RF	1	1	1
LOGREG	0.771	0.796	0.730

Table 5: Mean training set performance across trials over problems

Model	ADULT	COV	LETTER.P2	AVILA	BEAN
KNN	0.843	1.0	1.0	1.0	0.995
NN	0.781	0.918	1.0	0.881	0.950
RF	1.0	1.0	1.0	1.0	1.0
LOGREG	0.762	0.757	0.728	0.643	0.940

Table 6: Raw training performance (Ordered tuple by trials)

Dataset	KNN	NN
ADULT	(0.838,0.847,1,1,0.858)	(0.860,0.859,0.857,0.854,0.856)
COV	(1.0,1.0,1.0,1.0,1.0)	(0.931,0.929,0.922,0.923,0.934)
LETTER.P2	(1.0,1.0,1.0,1.0,1.0)	(1.0,1.0,1.0,1.0,0.999)
AVILA	(1.0,1.0,1.0,1.0,1.0)	(0.892,0.889,0.890,0.892,.889)
BEAN	(1.0,1.0,1.0,1.0,1.0)	(0.964,0.960,0.966,0.962,0.965)

Table 7: Raw training performance (Ordered tuple by trials)

Dataset	RF	LR
ADULT	(1.0,1.0,1.0,1.0,1.0)	(0.842,0.851,0.847,0.850,0.851)
COV	(1.0,1.0,1.0,1.0,1.0)	(0.745,0.763,0.753,0.757,0.767)
LETTER.P2	(1.0,1.0,1.0,1.0,1.0)	(0.736,0.727,0.721,0.730,0.720)
AVILA	(1.0,1.0,1.0,1.0,1.0)	(0.688,0.688,0.694,0.699,0.689)
BEAN	(1.0,1.0,1.0,1.0,1.0)	(0.955,0.958,0.958,0.956,0.959)

Table 8: Raw testing performance (Ordered tuple by trials)

Dataset	KNN	NN
ADULT	(0.834,0.831,0.834,0.832,0.831)	(0.850,0.849,0.848,0.851,0.849)
COV	(0.782,0.783,0.787,0.783,0.780)	(0.801,0.803,0.804,0.798,0.804)
LETTER.P2	(0.959,0.956,0.948,0.954,0.952)	(0.943,0.947,0.948,0.946,0.950)
AVILA	(0.803,0.804,0.803,0.801,0.788)	(0.836,0.838,0.841,0.838,0.831)
BEAN	(0.960,0.954,0.953,0.953,0.958)	(0.959,0.960,0.959,0.958,0.959)

Table 9: Raw testing performance (Ordered tuple by trials)

Dataset	RF	LR
ADULT	(0.842,0.850,0.846,0.847,0.846)	(0.846,0.846,0.846,0.843,0.847)
COV	(0.824,0.820,0.816,0.814,0.824)	(0.757,0.754,0.751,0.755,0.754)
LETTER.P2	(0.947,0.945,0.948,0.946,0.950)	(0.730,0.730,0.721,0.726,0.734)
AVILA	(0.975,0.966,0.958,0.972,0.970)	(0.685,0.684,0.685,0.685,0.681)
BEAN	(0.955,0.958,0.957,0.956,0.956)	(0.957,0.956,0.957,0.956,0.957)

References

Jock A. Blackard, Dr. Denis J. Dean, and Dr. Denis J. Dean. UCI machine learning repository, 1998. URL `https://archive.ics.uci.edu/ml/datasets/covertype`.

F. Fontanella A. ScottoÂ diÂ Freca C. DeÂ Stefano, M. Maniaci. Reliable writer identification in medieval manuscripts through page layout features: The 'avila' bible case. *Engineering Applications of Artificial Intelligence*, 72:99–110, 2018. URL `https://archive.ics.uci.edu/ml/datasets/Avila`.

Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, page 161–168, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933832. doi: 10.1145/1143844.1143865. URL `https://doi.org/10.1145/1143844.1143865`.

Ronny Kohavi and Barry Becker. UCI machine learning repository, 1994. URL `https://archive.ics.uci.edu/ml/datasets/adult`.

Murat Koklu and Ilker Ali Ozkan. Multiclass classification of dry beans using computer vision and machine learning techniques. *Computers and Electronics in Agriculture*, 174: 105507, 2020. ISSN 0168-1699. doi: https://doi.org/10.1016/j.compag.2020.105507. URL `https://www.sciencedirect.com/science/article/pii/S0168169919311573`.

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.

David J. Slate. UCI machine learning repository, 1991. URL `https://archive.ics.uci.edu/ml/datasets/Letter+Recognition`.

Project

March 18, 2021

[ ]:
```
%config InlineBackend.figure_format = 'retina'
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, cross_val_score,␣
 ↪GridSearchCV, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import make_scorer, accuracy_score, f1_score, roc_auc_score
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier

np.random.seed(42)

def encodeBind(df, features):
    dummies = pd.get_dummies(df[features])
    resDF = pd.concat([df, dummies], axis=1)
    resDF.drop(features, axis=1, inplace = True)
    return(resDF)

def preprocessDF(df, featureEncode, featureScale):
    scaler = StandardScaler()
    dfLabel = df.goal

    df.drop("goal", axis = 1, inplace = True)
    if featureEncode != None:
        df = encodeBind(df, featureEncode)
    df["goal"] = dfLabel

    df[featureScale] = scaler.fit_transform(df[featureScale])

    return(df)
```

```
[ ]: # Adult dataset
adultNames = ["age", "workclass", "fnlwgt", "education", "education-num",␣
 ↪"marital-status",
              "occupation", "relationship", "race", "sex", "capital-gain",␣
 ↪"capital-loss",
              "hours-per-week", "native-country", "goal"]
adultEncode = ["workclass", "education", "marital-status", "occupation",␣
 ↪"relationship", "race", "sex", "native-country"]
adultScale = ["age", "fnlwgt", "education-num", "capital-gain", "capital-loss",␣
 ↪"hours-per-week"]

adult = pd.read_table("adult.data", names = adultNames, sep = ",\s",␣
 ↪engine='python')
adult = adult[adult["workclass"] != "?"]
adult["goal"] = np.where(adult.goal == ">50K", 1, 0)
```

```
[ ]: adult[adultScale].hist()
```

```
[ ]: adult = preprocessDF(adult, adultEncode, adultScale)
adult
```

```
[ ]: # Covertype dataset
covNames = ["elevation", "aspect", "slope", "hordishydro", "verdishydro",␣
 ↪"hordisroad", "hillam", "hillnoon", "hillpm",
            "hordisfire"] + ["wild" + str(i) for i in range(1,5)] + ["soil" +␣
 ↪str(i) for i in range(1,41)] + ["goal"]
covScale = ["elevation", "aspect", "slope", "hordishydro", "verdishydro",␣
 ↪"hordisroad", "hillam", "hillnoon", "hillpm",
            "hordisfire"]
cov = pd.read_table("covtype.data", sep = ",", names = covNames)
cov.goal = np.where(cov.goal == cov["goal"].value_counts().idxmax(), 1, 0)
```

```
[ ]: cov[covScale].hist()
```

```
[ ]: cov = preprocessDF(cov, None, covScale)
cov
```

```
[ ]: # Letter dataset
letterNames = ["goal", "x-box", "y-box", "width", "height", "onpix", "x-bar",
              "y-bar", "x2bar", "y2bar", "xybar", "x2ybr", "xy2br", "x-ege",
              "xegvy", "y-ege", "yegvx"]
letterScale = ["x-box", "y-box", "width", "height", "onpix", "x-bar",
              "y-bar", "x2bar", "y2bar", "xybar", "x2ybr", "xy2br", "x-ege",
              "xegvy", "y-ege", "yegvx"]
letter = pd.read_table("letter-recognition.data", sep = ",", names =␣
 ↪letterNames)
letterCols = list(letter.columns)
```

```
letterCols[-1], letterCols[0] = letterCols[0], letterCols[-1]
letter = letter[letterCols]
chosenLetter = [chr(i) for i in range(ord('A'), ord('M')+1)]
letter.goal = letter.goal.apply(lambda x: 1 if x in chosenLetter else 0)
```

[ ]:
```
letter[letterScale].hist()
```

[ ]:
```
letter = preprocessDF(letter, None, letterScale)
letter
```

[ ]:
```
# Avila dataset
avilaNames = ["interdis", "upmar", "lowmar", "exploi", "rownum", "modratio",
              "interspace", "weight", "peaknum", "modratio-interspace", "goal"]
avila = pd.read_csv("avila.txt", sep = ",", names = avilaNames)
avila.goal = np.where(avila.goal == "A", 1, 0)
```

[ ]:
```
avila[list(avila.columns)[:-1]].hist()
```

[ ]:
```
avila
```

[ ]:
```
# Bean dataset
#beanNames = [""]
bean = pd.read_excel("Dry_Bean_Dataset.xlsx", engine='openpyxl')
beanScale = list(bean.columns)[:-1]
bean.Class = np.where(bean.Class == bean.Class.value_counts().idxmax(), 1, 0)
bean.columns = [*bean.columns[:-1], 'goal']
bean[beanScale].hist()
```

[ ]:
```
bean = preprocessDF(bean, None, beanScale)
bean
```

[ ]:
```
def getTrialTrainTest(pipe, param, score_function, X_train, X_test, Y_train,␣
 ↪Y_test):
    pipe.set_params(**param)
    pipe.fit(X_train, Y_train)
    y_pred_train = pipe.predict(X_train)
    y_pred_test = pipe.predict(X_test)
    trialTrain = score_function(Y_train, y_pred_train)
    trialTest = score_function(Y_test, y_pred_test)

    return trialTrain, trialTest

def spitoutres(data, pipe, param):
    aucTrialTrain = []
    accTrialTrain = []
    f1TrialTrain = []
    aucTrialTest = []
    accTrialTest = []
    f1TrialTest = []
    for trial in range(5):
        print(trial)
        X, Y = data.iloc[:,:-1], data.iloc[:,-1]
        X_train, X_test, Y_train, Y_test = train_test_split(X, Y,␣
 ↪train_size=5000, shuffle=True)
        gs = GridSearchCV(pipe, param_grid = param, cv = StratifiedKFold(),␣
 ↪n_jobs = -1,
                          scoring = scoring, refit = False)
        gs.fit(X_train, Y_train)
        results = gs.cv_results_

        auc_best_index = np.argmin(results['rank_test_AUC'])
        auc_best_param = results['params'][auc_best_index]

        accuracy_best_index = np.argmin(results['rank_test_Accuracy'])
        accuracy_best_param = results['params'][accuracy_best_index]

        f1_best_index = np.argmin(results['rank_test_F1'])
        f1_best_param = results['params'][f1_best_index]

        aucTr, aucTest = getTrialTrainTest(pipe, auc_best_param, roc_auc_score,␣
 ↪X_train, X_test, Y_train, Y_test)
        accTr, accTest = getTrialTrainTest(pipe, accuracy_best_param,␣
 ↪accuracy_score, X_train, X_test, Y_train, Y_test)
        f1Tr, f1Test = getTrialTrainTest(pipe, f1_best_param, f1_score,␣
 ↪X_train, X_test, Y_train, Y_test)


        aucTrialTrain.append(aucTr)
        accTrialTrain.append(accTr)
        f1TrialTrain.append(f1Tr)
        aucTrialTest.append(aucTest)
        accTrialTest.append(accTest)
        f1TrialTest.append(f1Test)

    return aucTrialTrain, accTrialTrain, f1TrialTrain, aucTrialTest,␣
 ↪accTrialTest, f1TrialTest

def runPerAlgo(algo, datasets, param):
    aucTrainMean, accTrainMean, f1TrainMean, aucTestMean, accTestMean,␣
 ↪f1TestMean = [], [], [], [], [], []
    aucTrainRaw, accTrainRaw, f1TrainRaw, aucTestRaw, accTestRaw, f1TestRaw =␣
 ↪[], [], [], [], [], []
    for data in datasets:
        aucTrain, accTrain, f1Train, aucTest, accTest, f1Test =␣
↪spitoutres(data, algo, param)

        accTrainRaw.append(accTrain)
        accTestRaw.append(accTest)

        aucTrainMean.append(np.mean(aucTrain))
        accTrainMean.append(np.mean(accTrain))
        f1TrainMean.append(np.mean(f1Train))
        aucTestMean.append(np.mean(aucTest))
        accTestMean.append(np.mean(accTest))
        f1TestMean.append(np.mean(f1Test))

    print("Raw train score")

    accTrainDF = pd.DataFrame(accTrainRaw, columns = ["Trial1", "Trial2",␣
↪"Trial3", "Trial 4", "Trial 5"], dtype = float)
    print(accTrainDF)

    print("Raw test score")

    accTestDF = pd.DataFrame(accTestRaw, columns = ["Trial1", "Trial2",␣
↪"Trial3", "Trial 4", "Trial 5"], dtype = float)
    print(accTestDF)

    theDFTest = pd.DataFrame(list(zip(aucTestMean, accTestMean, f1TestMean)),␣
↪columns = ["AUC", "ACC", "F1"], dtype = float)
    theDFTrain = pd.DataFrame(list(zip(aucTrainMean, accTrainMean,␣
↪f1TrainMean)), columns = ["AUC", "ACC", "F1"], dtype = float)

    print("Test over problems")
    print(theDFTest.mean(axis = 1))
    print("Test over metrics")
    print(theDFTest.mean(axis = 0))

    print("Train over problems")
    print(theDFTrain.mean(axis = 1))
    print("Train over metrics")
    print(theDFTrain.mean(axis = 0))
```

[ ]:
```
scoring = {'AUC': 'roc_auc', 'Accuracy': make_scorer(accuracy_score), 'F1':␣
↪make_scorer(f1_score)}

knnParam = {'knn__n_neighbors': np.arange(1,106,4),
    'knn__weights': ["uniform", "distance"]}
nnParam = [{
    "nn__hidden_layer_sizes": [(1,), (2,), (4,), (8,), (32,), (128,)],
    "nn__solver": ["adam"]
}, {
    "nn__hidden_layer_sizes": [(1,), (2,), (4,), (8,), (32,), (128,)],
    "nn__solver": ["sgd"],
    "nn__momentum": [0, 0.2, 0.5, 0.9]
}]
rfParam = {'rf__max_features': ["auto", "log2", None, 1, 2, 4, 6, 8, 10]}
lrParam = [{
    'lr__solver': ['saga'],
    'lr__penalty': ['l1', 'l2'],
    'lr__C': [10**i for i in range(-8,5)]
    }, {
    'lr__solver': ['lbfgs'],
    'lr__penalty': ['l2'],
    'lr__C': [10**i for i in range(-8,5)]
    }, {
    'lr__solver': ['lbfgs','saga'],
    'lr__penalty': ['none'],
}]

knnPipe = Pipeline([("knn", KNeighborsClassifier())])
nnPipe = Pipeline([("nn", MLPClassifier(max_iter=500))])
rfPipe = Pipeline([("rf", RandomForestClassifier(n_estimators = 1024, n_jobs =␣
 ↪-1))])
lrPipe = Pipeline([("lr", LogisticRegression(max_iter = 5000))])

params = [knnParam, nnParam, rfParam, lrParam]
pipes = [knnPipe, nnPipe, rfPipe, lrPipe]
datasets = [adult, cov, letter, avila, bean]
```

[ ]:
```
### KNN model
runPerAlgo(knnPipe, datasets, knnParam)
```

[ ]:
```
### NN model
runPerAlgo(nnPipe, datasets, nnParam)
```

[ ]:
```
### RF Model
runPerAlgo(rfPipe, datasets, rfParam)
```

[ ]:
```
### LR Model
runPerAlgo(lrPipe, datasets, lrParam)
```