Replication of Analysis performed by Caruana Niculescu-Mizil

Alan Krause ALKRAUSE@UCSD.EDU

Abstract

Many different papers have been published in various journals which compare classifiers against each other in the realm of machine learning. One of such papers was the analysis done by Caruana Niculescu-Mizil. In this report, I present a similar comparison as the one aforementioned, on three separate classifier models: Support Vector Machines, Logistic Regression, K-Nearest Neighbors, Random Forests, and Artificial Neural Networks.

1. Introduction

In Caruana Niculescu-Mizil, they write about STATLOG, which they call the "best known study" on comparing learning algorithms. Much in the same way that Caruana Niculescu-Mizil aimed to retest many of the same algorithms, I too will be implementing similar algorithms to the ones used, in the hopes of replicating the results and gaining more insight into the drawbacks of various algorithms.

While many different performance metrics can be used, at the most basic level we will be using the basic score of accuracy, where we simply check the percentage of correctly labeled points by our model. In order to accomplish this, all of the data that I use will have a binary label, that is a positive and negative class.

This paper will be comparing SVMs, Logistic Regression, KNN, Random Forests, and Artificial Neural Networks. For each classifier we are going to be tuning hyperparameters in order to get the best results, so that the performance of each of the classifiers will not have their results changed based on experimenter settings, thereby giving the best possible comparison between algorithms.

The results of the tests are what most would expect based off of the findings of Caruana Niculescu-Mizil. Random Forest and Support Vector Machines seem to have the greatest success across the datasets, whilst K-nearest neighbors and Logistic regression were the worst. Logistic Regression had some interesting results, where on some tasks it worked as well as SVMs and RFs, but was crushed on other datasets, suggesting that Logistic Regression is not as widely applicable as SVMs or RFs.

2. Methodology

2.1. Learning Algorithms

We look to go through many different hyperparameters in order to best represent the algorithms on the different datasets. This section will summarize all of the variations that we test the hyperparameters with (all of which are based off of the Caruana Niculescu-Mizil paper). All of the algorithms are trained through scikit learn.

Support Vector Machines (SVMS): We use linear, polynomial, and radial kernels, with the following hyper params. The polynomial kernel uses a second and third degree model, while the radial kernel is tested with the width

0.001,0.005,0.01,0.05,0.1,0.5,1,2. On each of the kernels we test the regularization parameter from $10^-7 to 10^3 by factor of ten$.

Logistic Regression (LOGREG): We train the model based on the l2 error function. We vary the regularization parameter by factors of 10 from $10^-8 to 10^4$, $as done in the CNM06 paper$.

K-Nearest Neighbors (KNN): We use KNN with k varying from 1 to 500 spaced log evenly and measure nearest neighbor using euclidean distance.

Random Forests (RF): We use Random Forests with a set number of 1024 trees. The size of the feature set considered at each split, in accordance with CNM06 is 1,2,4,6,8,12,15. (We only go to 15 here to fit the LETTERs dataset, which has the lowest number of features of all the tasks for the algorithms).

Artificial Neural Nets (ANN): We use sklearns implementation of the multilayer perceptron with max iterations set to 750. According to the implementation within CNM06 we vary the hidden layer sizes on sizes 1, 2, 4, 8, 32, 128 and set momentum to be one of the following: 0, .2, .5, .9.

2.2. Performance Metrics

As stated in the introduction, while Caruana Niculescu-Mizil use a large variety of different performance metrics, in order to keep the scope of the paper to a manageable level I will only be using an accuracy measure, where we base the performance of the model off of how many points in the data the classifier misclassified. Due to the preprocessing of the data that we have done, scoring based on binary classification will be an easy task.

2.3. Data Sets

Whereas Caruana Niculescu-Mizil compared the algorithms on 11 different classification problems, we will be comparing the classifiers on 3 of the 11, taken from the UCI data repository. ADULT is a binary predictor which takes 14 inputs and has a label on whether or not the selected adult makes ¿=50K annually or less. COVTYPE takes a variety of inputs on an area and maps each of the areas to a specific type of coverage, numbered 1-7. In order to create a binary classifier we set all 1's to 1 and the rest of the 2-7 will be set to 0. Finally LETTERS gives a variety of inputs and labels them as a letter. In order to create a binary classification problem we take the secondary method used in Caruana Niculescu-Mizil, where they set all letters A-M to the positive class, and the rest to the negative class.

In an effort to expand a bit more on the project and get more results on the algorithms, I also chose to include a few new datasets. One of which is the BANKS dataset from the UCI machine learning repository where we are given 21 columns of data on a customer at the bank and we aim to predict whether or not the client has subscribed to a term deposit. We choose this dataset because of its good categorical variables and its size, which is 45,212 clients. In addition to BANK we also add in DEFAULT, which gives us 25 attributes along with a binary classification on whether or not a client defaulted on their credit card payment. DEFAULT is yet another strong dataset from the UCI machine learning repository, with 30,000 separate entries.

3. Experiment

In order to train the models on the 3 separate problems, we take a random 5000 data points and train the model 5 times on different 5000 data points, using grid search to find the optimal parameters for each problem. Once the optimal params have been found, we take another 5000 data points and train the model

once more with the best parameters, then test in on the rest of the dataset that we left out. The testing score with optimal parameters is what we will report on.

The results generated are presented in Table 1 below, where a * indicates an insignificant difference between the top performing model and the score(p¡.05).

Looking at Table 1, we can notice a number of interesting discoveries. For one, Logistic Regression's poor overall score can be easily explained by its poor performance on the LETTERS task. However, looking at the ADULT task we see that Logistic regression outperformed K-nearest Neighbors, and even surpassed SVM in one particular case on the BANK task. Infact, if we were to remove the letters task then logistic regression would have outperformed KNN by mean score. This only goes to show that different models work better for different problems, and to get a better idea of overall performance we would need to test on a wider array of problems, as using 5 could be potentially unfair as if you perform poorly on one test it has a large negative effect on your mean score.

Beyond looking at the outliers, we can see that Random Forests were by far the best, getting the best accuracy score for 4 of the 5 datasets. They were closely followed by Support Vector Machines, and then Artificial Neural Networks. The bottom two were LOGREG and KNN.

Below we see the training scores generated by the models with their best hyperparams in Table 2.

Looking at Table 2 we can see that the models on average look to perform better on the overall training set by a small amount. LOGREG looked to perform the most consistently around its training set and test sets, followed by SVMs. RF and KNN are the worst in using the training score to determine test score, with RF perfectly fitting the training set every time and KNN twice. While this might look bad for RF, we must remember that on Table 1 we saw that despite the high training scores, it scored on average the best across all the datasets.

4. Conclusion

Looking at performance of the 5 models across the 5 tasks, we can see that SVM's and RF's appear to be more reliable than the rest of the classifiers, which one might expect given that it is relatively newer compared to KNN and LOGREG. It is worth noting that on some of the more computationally expensive datasets, ANN were limited to a maximum iterations of 750, which was not enough for the Multilayer Perceptron to converge. It is possible that with more time the ANN would have been able to find a better result and such move up to be closer with RFs and SVMs. Despite this ANNs managed to separate themselves from the bottom two algorithms. With such a small number of tasks that we trained the models on, there is a non zero chance that we may be misrepresenting the strength of the various models, as mentioned in the introduction. Despite that, I feel that we have given a decent summary of the performance of the various models across the datasets, as it ultimately matched up with the results published by Caruana Niculescu-Mizil.

5. Extra Credit

I believe that some of the work I have done here falls in the realm for extra credit. At the most basic level, I went out of my way to include 2 additional datasets, as well as implement another 2 classifiers. Due to my attention to making the project scalable, I was able to make this a bit easier to implement. In addition to the extra infrastructure to make scaling the project more feasible,

Table 1: Model Performance on Test Set

MODEL	ADULT	COV	LETTERS	BANK	DEFAULT	MEAN
RF	.8482	.8290	.9340	.9092	.8158	.8672
SVM	.8482	.8018	.9556	.9018*	.8136*	.8642
ANN	.8436	.8034	.9346	.8958*	.8112*	.85772
KNN	.8336	.7748	.9442	.8914	.8048	.84976
LOGREG	.8478*	.7598	.7220	.9016*	.8046	.80716

Table 2: Model Performance on Training Set

MODEL	ADULT	COV	LETTERS	BANK	DEFAULT	MEAN
LOGREG	.8540	.7780	.7298	.9066	.8140	+.00932
SVM	.8592	.8584	1	.9294	.8176	+.02872
ANN	.8342	.9342	1	.9130	.8280	+.04416
KNN	.8375	1	1	.9044	.8134	+.0613
RF	1	1	1	1	1	+.1238

this also increased the workload for my computer wholesale– putting in hours of work for some of the more computationally intensive datasets such as ADULT or DEFAULT.

In addition to the above, I also took the time to learn LaTex in order to properly present the paper in a standard way as outlined for the Journal of Machine Learning Research.

6. References

Caruana and Niculescu-Mizil (2006); Fleischer; Dua and Graff (2017); Pedregosa et al. (2011); Moro et al. (2014); Yeh and Lien (2009).

References

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

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

Jason Fleischer. Model selection.

Sérgio Moro, Paulo Cortez, and Paulo Rita. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 62:22–31, 2014.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

I-Cheng Yeh and Che-hui Lien. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 36(2):2473–2480, 2009.

Appendix A. Extra Tables

Table 3: P-Values of comparisons across algorithms for Table 1

MODEL	SVMs ADULT	RF COV	SVMs LETTERS	RF BANK	RF DEFAULT
LOGREG	.72314	.00016	.0000002	.087	.01884
KNN	.03552	.000354	.0221121	.002	.0366
SVMs	1.0	.0136	1.0	.1202	.6308
RF	.5979	1.0	.02847	1.0	1.0
ANN	.244	.036	.007904	.0892	.36716

Table 4: Raw 1-Run Test Scores at end of Param Fitting

MODEL	ADULT	COV	LETTERS	BANK	DEFAULT
LOGREG	.8481	.7665	.7289	.9014	.8092
KNN	.8276	.7957	.9498	.8916	.8052
SVMs	.8495	.8080	.9586	.9009	.8176
RF	.8540	.8364	.9462	.9018	.8177
ANN	.244	.036	.007904	.0892	.36716

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns; sns.set_style('white')  # plot formatting
from scipy import stats
##sklearn modules
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

## Manage our Data
adult = pd.read_csv("adult.data",header=None) ##index 14 = label
letter = pd.read_csv("letter-recognition.data", header=None) #index 0 = label
covtype = pd.read_csv("covtype.data", header=None) #index 54 = label
bank = pd.read_csv("bank.csv",header=None,sep=';',skiprows=1)
default = pd.read_excel("default.xls",skiprows=1)
##reassign lables according to Caruana Mizil
##covtype 7 is the positive class denoted by 1, and all else are denoted 0
covtype[54] = covtype[54].replace([2,3,4,5,6,7],0)
covtype[54] = covtype[54].replace(1,1)
##all letters a-m = 1 and others are 0
letter = letter.replace(['A','B','C','D','E','F','G','H','I','J','K','L','M'],1)
letter = letter.replace(['N','O','P','Q','R','S','T','U','V','W','X','Y','Z'],0)
##>50K == 1 , less than or equal == 0
##adult = adult.replace([" <=50K"," >50K"],[0,1])
adult = pd.get_dummies(adult)
##bank  column 52 == yes
bank = pd.get_dummies(bank)

##split data into X and Y
X = [adult.iloc[:,:107],covtype.iloc[:,:53],letter.iloc[:,1:],bank.iloc[:,:51], default.iloc[:,:23
Y = [adult.iloc[:,109],covtype.iloc[:,54],letter.iloc[:,0],bank.iloc[:,52], default.iloc[:,24]]

##Initialize the classifiers according to CM06
clf1 = LogisticRegression(multi_class='multinomial',
                          solver='newton-cg',
                          random_state=1)
clf2 = KNeighborsClassifier(algorithm='ball_tree',
                            leaf_size=50)
clf3 = SVC(random_state=1)

clf4 = RandomForestClassifier(n_estimators = 1024)

clf5 = MLPClassifier(max_iter = 750)

# Building the pipelines
pipe1 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf1)])

pipe2 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf2)])

pipe3 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf3)])

pipe4 = Pipeline([('std', StandardScaler()),
```

```
                  ('classifier', clf4)])

pipe5 = Pipeline([('std', StandardScaler()),
                  ('classifier', clf5)])

# Setting up the parameter grids according to CM06
param_grid1 = [{'classifier__penalty': ['l2'],
                'classifier__C': np.power(10., np.arange(-4, 8))}]

param_grid2 = [{'classifier__n_neighbors': [round(x) for x in list(np.logspace(np.log10(1), np.log
                'classifier__p': [2]}]  ##p == 2 means using euclidean distance

param_grid3 = [{'classifier__kernel': ['rbf'],
                'classifier__C': np.power(10., np.arange(-7, 3)),
                'classifier__gamma': list([0.001,0.005,0.01,0.05,0.1,0.5,1,2])},
               {'classifier__kernel': ['poly'],
                'classifier__C': np.power(10., np.arange(-7, 3)),
                'classifier__gamma': list([2,3])},
               {'classifier__kernel': ['linear'],
                'classifier__C': np.power(10., np.arange(-7, 3))}]

param_grid4 = [{'classifier__max_features': [1,2,4,6,8,12,15]}]

param_grid5 = [{'classifier__hidden_layer_sizes': [1,2,4,8,32,128],
                'classifier__momentum': [0,.2,.5,.9]}]

# Setting up multiple GridSearchCV objects, 1 for each algorithm
gridcvs = {}

for pgrid, est, name in zip((param_grid1, param_grid2, param_grid3, param_grid4, param_grid5),
                            (pipe1, pipe2, pipe3, pipe4, pipe5),
                            ('Logistic', 'KNN', 'SVM', 'RF', 'ANN')):
    gcv = GridSearchCV(estimator=est,
                       param_grid=pgrid,
                       scoring='accuracy',
                       n_jobs=-1,
                       cv=2, # just 2-fold inner loop, i.e. train/test
                       verbose=0,
                       refit=True)
    gridcvs[name] = gcv
```

In [ ]:

```
def testClassifiers(xData,yData,numTrials):

    for i in range(numTrials):
        print("Trial {}".format(i+1))
        X_train, X_test, y_train, y_test = train_test_split(xData,yData,train_size=5000,random_sta

        cv_scores = {name: [] for name, gs_est in gridcvs.items()}

        skfold = StratifiedKFold( n_splits=5, shuffle=True, random_state=i)

        # The outer loop for algorithm selection
        c = 1
        for outer_train_idx, outer_valid_idx in skfold.split(X_train,y_train):
            for name, gs_est in sorted(gridcvs.items()):
                print('outer fold %d/5 | 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)
```

```
            c += 1

    return cv_scores
```

In [ ]:
```
def GenerateSummary(name,xData,yData):
    print("Results for {}".format(name))
    best_algo = gridcvs[name]
    X_train, X_test, y_train, y_test = train_test_split(xData,yData,train_size=5000,random_state=6
    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('Accuracy %.2f%% (average over CV test folds)' %
          (100 * best_algo.best_score_))
    print('Best Parameters: %s' % gridcvs[name].best_params_)
    print('Training Accuracy: %.2f%%' % (100 * train_acc))
    print('Test Accuracy: %.2f%%' % (100 * test_acc))
```

In [ ]:
```
def TestDataset(nameData,x,y,numtrials):
    cv = testClassifiers(x,y,numtrials)

    bestMean = 0
    bestName = None

    print("\nResults on {} data\n".format(nameData))
    for name in cv:
        if np.mean(cv[name]) > bestMean:
            bestMean = np.mean(cv[name])
            bestName = name
        print('%-8s | outer CV acc. %.2f%% +\- %.3f' % (
              name, 100 * np.mean(cv[name]), 100 * np.std(cv[name])))
    print()
    for name in cv:
        print('{} best parameters'.format(name), gridcvs[name].best_params_)

    print()
    for name in cv:
        GenerateSummary(name,x,y)
        print('')

    for name in cv:
        t,p = stats.ttest_ind(cv[bestName],cv[name],equal_var=False)
        print('p:{} \nt:{} \nFor {}\nComparing against {}\n'.format(p,t,name,bestName))
```

In [ ]:
```
TestDataset("COV",X[1],Y[1],3)
TestDataset("LETTERS",X[2],Y[2],3)
TestDataset("ADULT",X[0],Y[0],3)
TestDataset("BANK",X[3],Y[3],3)
TestDataset("DEFAULT",X[4],Y[4],3)
```