A Comparison of Various Supervised Machine Learning Algorithms

Matthew Kang mjk095@ucsd.edu

COGS 118A, University of California San Diego

ABSTRACT

There are very few methodological evaluations of supervised machine learning algorithms from the modern era. This is in part, due to the fact that the expansion of the field of supervised machine learning is a relatively recent development. A paper done in 2006 by Caruana & Niculescu-Mizil elaborates on the variety of different supervised machine learning algorithms, and their respective accuracies. This paper seeks to accomplish a similar comparison of machine learning algorithms as that achieved in CNM06, on a smaller scale.

I. INTRODUCTION

The prevalence of data collection in daily life has become widespread in the modern era. Cell phones, laptops, smart TV's and other devices that people tend to use everyday serve as data collection entry points. From these things, there is now a plethora of data to sort through if one wishes to make algorithmic based predictions. Now that the problem of prediction has shifted from "how to collect enough data" to "how to utilize this data", the field of supervised machine learning has become an area of intense focus. An influx of medical data, for example, has led to the creation of many algorithms that can now predict whether a patient has a disease or not, using the contextual data of other medical patients with similar biological attributes. An example of a supervised machine learning algorithm that could be applicable in a situation like that would be K-Nearest Neighbors. Machine learning algorithms have paved the way from lidar technology for self driving cars to facial recognition. But not all algorithms are made equal. Some algorithms solve binary "0 or 1" prediction problems, while others give probabilities of events happening. Furthermore, different real life problems require different algorithms to solve them. In a 2006 paper by Caruana & Niculescu-Mizil, various different supervised machine learning algorithms are tested, compared, and analyzed. CNM06 tests ten different machine learning algorithms, using eight different performance metrics, on eleven different datasets. The validity of these datasets are then compiled and organized into a table. CNM06 has become an extremely influential paper in the field of supervised machine learning. In this paper, the procedures and methods of CNM06 will be replicated, in a simpler fashion. Three different supervised machine learning algorithms will be tested, on three different data sets. The only performance metric being accounted for will be accuracy.

II. METHODOLOGY

2.1 Algorithms Used

Listed below are the three algorithms that will be used and compared throughout this paper. The parameters and parameter specifications will be the same as those in the CNM06 paper. Specific hyperparameters will be tested for and found. Each three algorithms will be tested on three different datasets, for three trials. That means there will be 27 total trials. Each trial will randomly choose 5000 data points within the respective dataset for five cross validation in order to find hyperparameters via gridsearch.

K-Nearest Neighbors :

KNN will be implemented. Distance between points will be measured by the Euclidean Distance. The size of the training set will be 25 k values used. Hyperparameters will be optimized using gridsearch.

Support Vector Machines:

The regularization parameter, C, will vary by factors of ten from 10^-7 to 10^3.

Hyperparameters will be optimized using gridsearch.

Logistic Regression:

The hyperparameter for ridge regression (lambda) will be tested by factors of 10 from 10^-8 to 10^4. This would indicate 14 different hyperparameter settings being tested.

Hyperparameters will be optimized using gridsearch.

2.2 Performance Metrics

The only performance metric being used in this paper will be accuracy. This is the ratio of the number of predicted elements that exist within the set of the true elements. It is accessed using sklearn.metrics.accuracy_score.

2.3 Datasets

Three datasets will be used. They have all been retrieved from the UCI Machine Learning Repository.

Adults

The adult data set represents a variety of census data on around 50,000 adults. The original intended purpose of this data set was to find out if there were factors that would determine whether or not an adult would make more than $50,000 a year. Features include things like age, employment status, race, education, marital status,sex, occupation, country of origin, etc.

Bank

The bank marketing data set represents bank data gathered from around 50,000 adults. It's original intended purpose was for use with bank telemarketing. Each row represents one client of the bank. Things like age, income, occupation, marital status, and loan status are recorded to name a few features.

Cov_type

The cover type data set is from a geological study. It tracks data related to forest cover. 30 meter by 30 meter squares of forest were analyzed in northern Colorado, with features being listed for each square. Things like elevation, slope, soil type, and cover type were recorded, to name a few. There are 40 different soil types, with each soil type being one-hot encoded. I cannot stress how difficult that made things.

Experiment

Process

The basics of this experiment are as follows. We have our three datasets, adult, bank, and cov_type. On each of these three datasets, we will perform KNN, SVM, and Linear regression. Each of these algorithms will be performed three times. So on adult, for example, we will perform KNN, SVM, and Linear regression. Adult KNN will be performed for three trials, adult SVM will be performed for three trials, adult Linear regression will be performed for three trails. Same process applies for bank and cov_type.We will search for hyperparameters through gridsearch optimization. Each algorithm (KNN,SVM,LinReg) will have a different number of hyperparameter settings. In order to find the optimal hyperparameters, 5 fold cross validation will be used on 5,000 randomly selected points. Once the optimal hyperparameters have been chosen, the model will be tested on all the remaining points. (minus the 5000 that were used for hyperparameter optimization).

Table 1 : Mean Test Set Performance for each Algorithm/Dataset Combination

	Mean Accuracy : Adult	Mean Accuracy: Bank	Mean Accuracy for Cov_type
KNN	0.8214	0.8847	incomplete
SVM	0.7867	0.7733	incomplete
LinReg	0.8058	0.88525	incomplete

The best hyperparameters for KNN for the adult data was a k value of 4 with uniform weights.

The best hyperparameters for KNN for the bank data was a k value of 8 with uniform weights.

The best hyperparameters for SVM for the adult data was a C value of 0.1 with a linear kernel.

The best hyperparameters for SVM for the bank data was a C value of 10^-7 with a linear kernel.

The best hyperparameters for LinReg for the adult data was a C value of 1 with penalty being L2.

The best hyperparameters for LinReg for the adult data was a C value of 10^-8 with penalty being L2.

Conclusion

Different algorithms have different hyperparameters with much testing needing to be done in order to find the optimal hyperparameters. The abundance of algorithms that exist today with which programmers and analysts can apply to data means that drawing conclusions from data is now a matter of choosing the right algorithm. Different algorithms for different situations. This paper, in its attempts to replicate the 2006 paper by Caruna and Niculescu, has showed that different supervised machine learning algorithms will perform either better or worse when faced with different data and hyperparameters.

References

Caruana, Rich., & Niculescu-Mizil , Alexandru. *An Empirical Comparison of Supervised Learning Algorithms.* Department of Computer Science, Cornell University, Ithaca

Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

In [55]:
```
import pandas as pd
import numpy as np
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV, cross_v
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.linear_model import LogisticRegression
```

In [5]:
```
#Data Cleaning and preprocessing
adult_preprocessed = pd.read_csv('adult.data',names = ["age","workclass","f
adult_preprocessed.dropna(inplace=True)


bank_preprocessed = pd.read_csv("bank-full.csv", sep = ';')

cov_type_preprocessed=pd.read_csv('covtype.data.gz',names = ['Elevation', '
        'Vertical_Distance_To_Hydrology', 'Horizontal_Distance_To_Roadways',
        'Hillshade_9am', 'Hillshade_Noon', 'Hillshade_3pm',
        'Horizontal_Distance_To_Fire_Points', 'Wilderness_Area1',
        'Wilderness_Area2', 'Wilderness_Area3', 'Wilderness_Area4',
        'Soil_Type1', 'Soil_Type2', 'Soil_Type3', 'Soil_Type4', 'Soil_Type5'
        'Soil_Type6', 'Soil_Type7', 'Soil_Type8', 'Soil_Type9', 'Soil_Type10
        'Soil_Type11', 'Soil_Type12', 'Soil_Type13', 'Soil_Type14',
        'Soil_Type15', 'Soil_Type16', 'Soil_Type17', 'Soil_Type18',
        'Soil_Type19', 'Soil_Type20', 'Soil_Type21', 'Soil_Type22',
        'Soil_Type23', 'Soil_Type24', 'Soil_Type25', 'Soil_Type26',
        'Soil_Type27', 'Soil_Type28', 'Soil_Type29', 'Soil_Type30',
        'Soil_Type31', 'Soil_Type32', 'Soil_Type33', 'Soil_Type34',
        'Soil_Type35', 'Soil_Type36', 'Soil_Type37', 'Soil_Type38',
        'Soil_Type39', 'Soil_Type40', 'Cover_Type'])
```

In [6]:
```
#Useable data. 10,000 random samples are used from each data set. This is t
Adult = adult_preprocessed.sample(10000,random_state=1)
Adult['income']=[1 if each==' >50K' else 0 for each in Adult['income']]


Bank = bank_preprocessed.sample(10000,random_state=1)

Cov_type = cov_type_preprocessed.sample(10000,random_state=1)
```

In [7]:
```
#ADULT : PHASE 1
#The only things I want to use from the adult data set are the numeric valu
nAdult = Adult[['age','educational-num','capital-gain','capital-loss','hour
```

In [360]:

```
#ADULT
#KNN
#TRIAL 1
AKNN1 = nAdult.sample(5000,random_state=1)
AKNN1data = AKNN1.iloc[:,0:5].values
AKNN1target = AKNN1.income.values
knn = KNeighborsClassifier()
param_grid = dict(n_neighbors = list(range(1,25)), weights = ["uniform", "d
grid = GridSearchCV(knn,param_grid, cv = 5, scoring = 'accuracy')
grid.fit(AKNN1data,AKNN1target)

print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
```

```
0.8231999999999999
{'n_neighbors': 12, 'weights': 'uniform'}
KNeighborsClassifier(n_neighbors=12)
```

In [ ]:

In [347]:

```
#ADULT
#KNN
#TRIAL 2
AKNN2 = nAdult.sample(5000,random_state=2)
AKNN2data = AKNN2.iloc[:,0:5].values
AKNN2target = AKNN2.income.values
knn = KNeighborsClassifier()
param_grid = dict(n_neighbors = list(range(1,25)), weights = ["uniform", "d
grid = GridSearchCV(knn,param_grid, cv = 5, scoring = 'accuracy')
grid.fit(AKNN2data,AKNN2target)

print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
```

```
0.8235999999999999
{'n_neighbors': 4, 'weights': 'uniform'}
KNeighborsClassifier(n_neighbors=4)
```

In [ ]:

In [348]:

```
#ADULT
#KNN
#TRIAL 3
AKNN3 = nAdult.sample(5000,random_state=3)
AKNN3data = AKNN3.iloc[:,0:5].values
AKNN3target = AKNN3.income.values
knn = KNeighborsClassifier()
param_grid = dict(n_neighbors = list(range(1,25)), weights = ["uniform", "d
grid = GridSearchCV(knn,param_grid, cv = 5, scoring = 'accuracy')
grid.fit(AKNN3data,AKNN3target)

print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
```

```
0.8173999999999999
{'n_neighbors': 6, 'weights': 'uniform'}
KNeighborsClassifier(n_neighbors=6)
```

In [19]:

```
#SVM
#ADULT
#TRIAL 1
ASVM1 = nAdult.sample(5000,random_state=1)
ASVM1_X = ASVM1.iloc[:,0:5].values
ASVM1_Y = ASVM1.income.values
```

In [35]:

```
X_train, X_test, y_train, y_test = train_test_split(ASVM1_X, ASVM1_Y, test_
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

In [51]:

```
classifier_svm = svm.SVC(kernel = 'linear')
C_list = [0.0000001, 0.000001, 0.00001, 0.0001, 0.001, 0.01,0.1,1,10,100]
grid1 = GridSearchCV(classifier_svm, {'C':C_list}, scoring='accuracy', cv =
grid1.fit(X_train, y_train)
clf_svm = svm.SVC(C = grid1.best_params_['C'], kernel = 'linear')
clf_svm.fit(X_train, y_train)
pred_svm = clf_svm.predict(X_test)
print("Support Vector Machine: ", classification_report(y_test, pred_svm))
```

```
Support Vector Machine:                precision    recall  f1-score   su
pport

           0       0.80      0.96      0.87       752
           1       0.69      0.27      0.39       248

    accuracy                           0.79      1000
   macro avg       0.74      0.62      0.63      1000
weighted avg       0.77      0.79      0.75      1000
```

In [53]:

```
#TRIAL 2
ASVM2 = nAdult.sample(5000,random_state=2)
ASVM2_X = ASVM2.iloc[:,0:5].values
ASVM2_Y = ASVM2.income.values

X_train2, X_test2, y_train2, y_test2 = train_test_split(ASVM2_X, ASVM2_Y, t
sc = StandardScaler()
X_train2 = sc.fit_transform(X_train2)
X_test2 = sc.fit_transform(X_test2)

classifier_svm = svm.SVC(kernel = 'linear')
grid2 = GridSearchCV(classifier_svm, {'C':C_list}, scoring='accuracy', cv =
grid2.fit(X_train2, y_train2)
clf_svm2 = svm.SVC(C = grid2.best_params_['C'], kernel = 'linear')
clf_svm2.fit(X_train2, y_train2)
pred_svm2 = clf_svm2.predict(X_test2)
print("Support Vector Machine: ", classification_report(y_test2, pred_svm2)
```

```
Support Vector Machine:                precision    recall  f1-score   su
pport

           0       0.80      0.96      0.88       754
           1       0.70      0.28      0.40       246

    accuracy                           0.79      1000
   macro avg       0.75      0.62      0.64      1000
weighted avg       0.78      0.79      0.76      1000
```

In [49]:

```
#TRIAL 3
ASVM3 = nAdult.sample(5000,random_state=3)
ASVM3_X = ASVM3.iloc[:,0:5].values
ASVM3_Y = ASVM3.income.values

X_train3, X_test3, y_train3, y_test3 = train_test_split(ASVM3_X, ASVM3_Y, t
sc = StandardScaler()
X_train3 = sc.fit_transform(X_train3)
X_test3 = sc.fit_transform(X_test3)

classifier_svm = svm.SVC(kernel = 'linear')
grid3 = GridSearchCV(classifier_svm, {'C':C_list}, scoring='accuracy', cv =
grid3.fit(X_train3, y_train3)
clf_svm3 = svm.SVC(C = grid3.best_params_['C'], kernel = 'linear')
clf_svm3.fit(X_train3, y_train3)
pred_svm3 = clf_svm3.predict(X_test3)
print("Support Vector Machine: ", classification_report(y_test3, pred_svm3)
```

```
Support Vector Machine:                precision    recall  f1-score   su
pport

           0       0.81      0.97      0.89       756
           1       0.79      0.31      0.44       244

    accuracy                           0.81      1000
   macro avg       0.80      0.64      0.66      1000
weighted avg       0.81      0.81      0.78      1000

0.8005000000000001
{'C': 0.1}
SVC(C=0.1, kernel='linear')
```

In [54]:

```
#SVM RESULTS
print (grid1.best_score_)
print (grid1.best_params_)
print (grid1.best_estimator_)

print (grid2.best_score_)
print (grid2.best_params_)
print (grid2.best_estimator_)

print (grid3.best_score_)
print (grid3.best_params_)
print (grid3.best_estimator_)
```

```
0.7927500000000001
{'C': 0.1}
SVC(C=0.1, kernel='linear')
0.7987499999999998
{'C': 0.1}
SVC(C=0.1, kernel='linear')
0.8005000000000001
{'C': 0.1}
SVC(C=0.1, kernel='linear')
```

In [83]:

```
#ADULT
#LIN REG
#TRIAL 1

LR1 = nAdult.sample(5000,random_state=10)

LR1_X = LR1.iloc[:,0:5].values
LR1_Y = LR1.income.values

LR1X_train, LR1X_test, LR1y_train, LR1y_test = train_test_split(LR1_X, LR1_
sc = StandardScaler()
LR1X_train = sc.fit_transform(LR1X_train)
LR1X_test = sc.fit_transform(LR1X_test)




LRgrid={"C":np.logspace(-8,4,13), "penalty":["l1","l2"]}
logreg=LogisticRegression()
logreg_cv=GridSearchCV(logreg,LRgrid,cv=5)
logreg_cv.fit(LR1X_train,LR1y_train)

print("tuned hyperparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

```
tor fit failed. The score on this train-test partition for these paramete
rs will be set to nan. Details:
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/s
ite-packages/sklearn/model_selection/_validation.py", line 531, in _fit_a
nd_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/s
ite-packages/sklearn/linear_model/_logistic.py", line 1304, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/s
ite-packages/sklearn/linear_model/_logistic.py", line 443, in _check_solv
er
    "got %s penalty." % (solver, penalty))
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 p
enalty.

  FitFailedWarning)
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-pack
ages/sklearn/model_selection/_validation.py:552: FitFailedWarning: Estima
```

In [84]:

```
#ADULT
#LIN REG
#TRIAL 2

LR2 = nAdult.sample(5000,random_state=20)

LR2_X = LR2.iloc[:,0:5].values
LR2_Y = LR2.income.values

LR2X_train, LR2X_test, LR2y_train, LR2y_test = train_test_split(LR2_X, LR2_
sc = StandardScaler()
LR2X_train = sc.fit_transform(LR2X_train)
LR2X_test = sc.fit_transform(LR2X_test)




LRgrid2={"C":np.logspace(-8,4,13), "penalty":["l1","l2"]}
logreg2=LogisticRegression()
logreg_cv2=GridSearchCV(logreg2,LRgrid2,cv=5)
logreg_cv2.fit(LR2X_train,LR2y_train)

print("tuned hyperparameters :(best parameters) ",logreg_cv2.best_params_)
print("accuracy :",logreg_cv2.best_score_)
```

```
  File "/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/s
ite-packages/sklearn/linear_model/_logistic.py", line 1304, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/s
ite-packages/sklearn/linear_model/_logistic.py", line 443, in _check_solv
er
    "got %s penalty." % (solver, penalty))
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 p
enalty.

  FitFailedWarning)
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-pack
ages/sklearn/model_selection/_validation.py:552: FitFailedWarning: Estima

tor fit failed. The score on this train-test partition for these paramete
rs will be set to nan. Details:
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/s
ite-packages/sklearn/model_selection/_validation.py", line 531, in _fit_a
nd_score
```

In [85]:

```
#ADULT
#LIN REG
#TRIAL 3

LR3 = nAdult.sample(5000,random_state=30)

LR3_X = LR3.iloc[:,0:5].values
LR3_Y = LR3.income.values

LR3X_train, LR3X_test, LR3y_train, LR3y_test = train_test_split(LR3_X, LR3_
sc = StandardScaler()
LR3X_train = sc.fit_transform(LR3X_train)
LR3X_test = sc.fit_transform(LR3X_test)




LRgrid3={"C":np.logspace(-8,4,13), "penalty":["l1","l2"]}
logreg3=LogisticRegression()
logreg_cv3=GridSearchCV(logreg3,LRgrid3,cv=5)
logreg_cv3.fit(LR3X_train,LR3y_train)

print("tuned hpyerparameters :(best parameters) ",logreg_cv3.best_params_)
print("accuracy :",logreg_cv3.best_score_)
```

```
ite-packages/sklearn/model_selection/_validation.py", line 531, in _fit_a
nd_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/s
ite-packages/sklearn/linear_model/_logistic.py", line 1304, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/s
ite-packages/sklearn/linear_model/_logistic.py", line 443, in _check_solv
er
    "got %s penalty." % (solver, penalty))
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 p
enalty.

  FitFailedWarning)
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-pack
ages/sklearn/model_selection/_validation.py:552: FitFailedWarning: Estima

tor fit failed. The score on this train-test partition for these paramete
rs will be set to nan. Details:
Traceback (most recent call last):
```

In [115]:

```
#BANK
#PHASE 1
#KNN

Bank['binary_y'] = Bank.y.apply(lambda x: 0 if x == 'no' else 1)

nBank = Bank.sample(5000,random_state=1)[['age','balance','binary_y']]
```

In [116]:

```
nBank_X = nBank.iloc[:,0:2]
nBank_Y = nBank.binary_y
```

In [117]:

```
knn = KNeighborsClassifier()
param_grid = dict(n_neighbors = list(range(1,25)), weights = ["uniform", "d
B_grid = GridSearchCV(knn,param_grid, cv = 5, scoring = 'accuracy')
B_grid.fit(nBank_X,nBank_Y)

print (B_grid.best_score_)
print (B_grid.best_params_)
print (B_grid.best_estimator_)
```

```
0.8817999999999999
{'n_neighbors': 15, 'weights': 'uniform'}
KNeighborsClassifier(n_neighbors=15)
```

In [ ]:

In [118]:

```
nBank2 = Bank.sample(5000,random_state=2)[['age','balance','binary_y']]
```

In [119]:

```
B2_grid = GridSearchCV(knn,param_grid, cv = 5, scoring = 'accuracy')
B2_grid.fit(nBank2.iloc[:,0:2],nBank2.binary_y)

print (B2_grid.best_score_)
print (B2_grid.best_params_)
print (B2_grid.best_estimator_)
```

```
0.8876000000000002
{'n_neighbors': 8, 'weights': 'uniform'}
KNeighborsClassifier(n_neighbors=8)
```

In [121]:

```
nBank3 = Bank.sample(5000,random_state=3)[['age','balance','binary_y']]
B3_grid = GridSearchCV(knn,param_grid, cv = 5, scoring = 'accuracy')
B3_grid.fit(nBank3.iloc[:,0:2],nBank3.binary_y)

print (B3_grid.best_score_)
print (B3_grid.best_params_)
print (B3_grid.best_estimator_)
```

```
0.8847999999999999
{'n_neighbors': 10, 'weights': 'uniform'}
KNeighborsClassifier(n_neighbors=10)
```

In [128]:

```
#BANK
#SVM

BSVM1 = Bank.sample(5000,random_state=100)[['age','balance','binary_y']]
BSVM1_X = BSVM1.iloc[:,0:2]
BSVM1_Y = BSVM1.binary_y

BX_train, BX_test, By_train, By_test = train_test_split(BSVM1_X, BSVM1_Y, t
sc = StandardScaler()
BX_train = sc.fit_transform(BX_train)
BX_test = sc.fit_transform(BX_test)

classifier_svm = svm.SVC(kernel = 'linear')
C_list = [0.0000001, 0.000001, 0.00001, 0.0001, 0.001, 0.01,0.1,1,10,100]
Bgrid1 = GridSearchCV(classifier_svm, {'C':C_list}, scoring='accuracy', cv
Bgrid1.fit(BX_train, By_train)
Bclf_svm = svm.SVC(C = Bgrid1.best_params_['C'], kernel = 'linear')
Bclf_svm.fit(BX_train, By_train)
Bpred_svm = Bclf_svm.predict(BX_test)
print("Support Vector Machine: ", classification_report(By_test, Bpred_svm)

print (Bgrid1.best_score_)
print (Bgrid1.best_params_)
print (Bgrid1.best_estimator_)
```

```
Support Vector Machine:                precision    recall  f1-score   su
pport

           0       0.88      1.00      0.94       881
           1       0.00      0.00      0.00       119

    accuracy                           0.88      1000
   macro avg       0.44      0.50      0.47      1000
weighted avg       0.78      0.88      0.83      1000

0.89075
{'C': 1e-07}
SVC(C=1e-07, kernel='linear')

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-pack
ages/sklearn/metrics/_classification.py:1221: UndefinedMetricWarning: Pre
cision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavio
r.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [130]:
```
BSVM2 = Bank.sample(5000,random_state=200)[['age','balance','binary_y']]
BSVM2_X = BSVM2.iloc[:,0:2]
BSVM2_Y = BSVM2.binary_y

B2X_train, B2X_test, B2y_train, B2y_test = train_test_split(BSVM2_X, BSVM2_
sc = StandardScaler()
B2X_train = sc.fit_transform(B2X_train)
B2X_test = sc.fit_transform(B2X_test)

C_list = [0.0000001, 0.000001, 0.00001, 0.0001, 0.001, 0.01,0.1,1,10,100]
Bgrid2 = GridSearchCV(classifier_svm, {'C':C_list}, scoring='accuracy', cv
Bgrid2.fit(B2X_train, B2y_train)
B2clf_svm = svm.SVC(C = Bgrid2.best_params_['C'], kernel = 'linear')
B2clf_svm.fit(B2X_train, B2y_train)
B2pred_svm = B2clf_svm.predict(B2X_test)
print("Support Vector Machine: ", classification_report(B2y_test, B2pred_sv

print (Bgrid2.best_score_)
print (Bgrid2.best_params_)
print (Bgrid2.best_estimator_)
```

```
Support Vector Machine:                precision    recall  f1-score   su
pport

           0       0.87      1.00      0.93       874
           1       0.00      0.00      0.00       126

    accuracy                           0.87      1000
   macro avg       0.44      0.50      0.47      1000
weighted avg       0.76      0.87      0.82      1000

0.8875
{'C': 1e-07}
SVC(C=1e-07, kernel='linear')

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-pack
ages/sklearn/metrics/_classification.py:1221: UndefinedMetricWarning: Pre
cision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavio
r.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [131]:

```
BSVM3 = Bank.sample(5000,random_state=300)[['age','balance','binary_y']]
BSVM3_X = BSVM3.iloc[:,0:2]
BSVM3_Y = BSVM3.binary_y

B3X_train, B3X_test, B3y_train, B3y_test = train_test_split(BSVM3_X, BSVM3_
sc = StandardScaler()
B3X_train = sc.fit_transform(B3X_train)
B3X_test = sc.fit_transform(B3X_test)

C_list = [0.0000001, 0.000001, 0.00001, 0.0001, 0.001, 0.01,0.1,1,10,100]
Bgrid3 = GridSearchCV(classifier_svm, {'C':C_list}, scoring='accuracy', cv
Bgrid3.fit(B3X_train, B3y_train)
B3clf_svm = svm.SVC(C = Bgrid3.best_params_['C'], kernel = 'linear')
B3clf_svm.fit(B3X_train, B3y_train)
B3pred_svm = B3clf_svm.predict(B3X_test)
print("Support Vector Machine: ", classification_report(B3y_test, B3pred_sv

print (Bgrid3.best_score_)
print (Bgrid3.best_params_)
print (Bgrid3.best_estimator_)
```

```
Support Vector Machine:                precision    recall  f1-score   su
pport

           0       0.88      1.00      0.94       882
           1       0.00      0.00      0.00       118

    accuracy                           0.88      1000
   macro avg       0.44      0.50      0.47      1000
weighted avg       0.78      0.88      0.83      1000

0.8835000000000001
{'C': 1e-07}
SVC(C=1e-07, kernel='linear')

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-pack
ages/sklearn/metrics/_classification.py:1221: UndefinedMetricWarning: Pre
cision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavio
r.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [134]:

```
#BANK
#LIN REG
#1

BLR1 = Bank.sample(5000,random_state=11)[['age','balance','binary_y']]

BLR1_X = BLR1.iloc[:,0:2]
BLR1_Y = BLR1.binary_y

BLR1X_train, BLR1X_test, BLR1y_train, BLR1y_test = train_test_split(BLR1_X,
sc = StandardScaler()
BLR1X_train = sc.fit_transform(BLR1X_train)
BLR1X_test = sc.fit_transform(BLR1X_test)




BLRgrid={"C":np.logspace(-8,4,13), "penalty":["l1","l2"]}
logreg=LogisticRegression()
Blogreg_cv=GridSearchCV(logreg,BLRgrid,cv=5)
Blogreg_cv.fit(BLR1X_train,BLR1y_train)

print("tuned hyperparameters :(best parameters) ",Blogreg_cv.best_params_)
print("accuracy :",Blogreg_cv.best_score_)
```

```
  File "/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/s
ite-packages/sklearn/linear_model/_logistic.py", line 1304, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/s
ite-packages/sklearn/linear_model/_logistic.py", line 443, in _check_solv
er
    "got %s penalty." % (solver, penalty))
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 p
enalty.

  FitFailedWarning)
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-pack
ages/sklearn/model_selection/_validation.py:552: FitFailedWarning: Estima

tor fit failed. The score on this train-test partition for these paramete
rs will be set to nan. Details:
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/s
ite-packages/sklearn/model_selection/_validation.py", line 531, in _fit_a
nd_score
    estimator.fit(X_train, y_train, **fit_params)
```

In [135]:
```
#BANK
#LIN REG
#2

BLR2 = Bank.sample(5000,random_state=22)[['age','balance','binary_y']]

BLR2_X = BLR2.iloc[:,0:2]
BLR2_Y = BLR2.binary_y

BLR2X_train, BLR2X_test, BLR2y_train, BLR2y_test = train_test_split(BLR2_X,
sc = StandardScaler()
BLR2X_train = sc.fit_transform(BLR2X_train)
BLR2X_test = sc.fit_transform(BLR2X_test)




BLR2grid={"C":np.logspace(-8,4,13), "penalty":["l1","l2"]}

B2logreg_cv=GridSearchCV(logreg,BLR2grid,cv=5)
B2logreg_cv.fit(BLR2X_train,BLR2y_train)

print("tuned hyperparameters :(best parameters) ",B2logreg_cv.best_params_)
print("accuracy :",B2logreg_cv.best_score_)
```

```
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/s
ite-packages/sklearn/model_selection/_validation.py", line 531, in _fit_a
nd_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/s
ite-packages/sklearn/linear_model/_logistic.py", line 1304, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/s
ite-packages/sklearn/linear_model/_logistic.py", line 443, in _check_solv
er
    "got %s penalty." % (solver, penalty))
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 p
enalty.

  FitFailedWarning)
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-pack
ages/sklearn/model_selection/_validation.py:552: FitFailedWarning: Estima
tor fit failed. The score on this train-test partition for these paramete
rs will be set to nan. Details:
Traceback (most recent call last):
```

In [136]:

```
#BANK
#LIN REG
#3

BLR3 = Bank.sample(5000,random_state=33)[['age','balance','binary_y']]

BLR3_X = BLR3.iloc[:,0:2]
BLR3_Y = BLR3.binary_y

BLR3X_train, BLR3X_test, BLR3y_train, BLR3y_test = train_test_split(BLR3_X,
sc = StandardScaler()
BLR3X_train = sc.fit_transform(BLR3X_train)
BLR3X_test = sc.fit_transform(BLR3X_test)




BLR3grid={"C":np.logspace(-8,4,13), "penalty":["l1","l2"]}

B3logreg_cv=GridSearchCV(logreg,BLR3grid,cv=5)
B3logreg_cv.fit(BLR3X_train,BLR3y_train)

print("tuned hyperparameters :(best parameters) ",B3logreg_cv.best_params_)
print("accuracy :",B3logreg_cv.best_score_)
```

```
rs will be set to nan. Details:
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/s
ite-packages/sklearn/model_selection/_validation.py", line 531, in _fit_a
nd_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/s
ite-packages/sklearn/linear_model/_logistic.py", line 1304, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/s
ite-packages/sklearn/linear_model/_logistic.py", line 443, in _check_solv
er
    "got %s penalty." % (solver, penalty))
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 p
enalty.

  FitFailedWarning)
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-pack
ages/sklearn/model_selection/_validation.py:552: FitFailedWarning: Estima
tor fit failed. The score on this train-test partition for these paramete
```

In [ ]: