Joseph Marquez

jvm2@njit.edu

Professor Yasser Abduallah

CS634 104

Github Link: https://github.com/jvm2NJIT/CS634-FinalProject

Final Project Report

File Overview

data folder: Folder that contains the data used in this project

> **bank-full.csv**: Slightly edited version of bank-full-original.csv
>
> **bank-full-original.csv**: Data used in this project, originally from this site:
>
> https://archive.ics.uci.edu/dataset/222/bank+marketing

part1_randomforest.ipynb: My code for the Random Forest implementation

part2_knn.ipynb: My code for the KNN implementation

part3_lstm.ipynb: My code for the LSTM implementation

FinalReport.pdf: Report of the project

Data Overview

My dataset came from the UC Irvine Machine Learning Repository. To quote the site, “The data is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict if the client will subscribe a term deposit (variable y).” There are 20 variables in the dataset, including categorical, boolean, date, and numerical variables. There are also 45211 lines of data. For the purpose of this project, I only used 7 of these features in my models. The features I used are as follows:

1. Age
2. Marital
    a. Divorced mapped to 0
    b. Single and Unknown mapped to 1
    c. Married mapped to 2
3. Default
    a. No and Unknown mapped to 0
    b. Yes mapped to 1
4. Housing
    a. No and Unknown mapped to 0
    b. Yes mapped to 1
5. Loan
    a. No and Unknown mapped to 0
    b. Yes mapped to 1
6. Cons.price.idx

7. Cons.conf.idx

The Y column also had values of No mapped to 0 and values of Yes mapped to 1.

Model Overview

For each of the 3 machine learning methods, I used existing Python libraries to create the model. Then I used 10-fold validation, tested my model for each fold, and recorded the performance metrics of each fold as well. Afterwards, I took the average of the performance metrics over these 10 folds to get a good indication of how successful the models were.

Random Forest

```
kf = KFold(n_splits=10)
rf = RandomForestClassifier()
classifier_performance = []

for i, (train_indices, test_indices) in enumerate(kf.split(X)):
    X_train, y_train = X.iloc[train_indices], y.iloc[train_indices]
    X_test, y_test = X.iloc[test_indices], y.iloc[test_indices]

    rf.fit(X_train, y_train)

    y_pred = rf.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    TN, FP, FN, TP = cm.ravel()
    # TP = cm[0][0]
    # FN = cm[1][0]
    # TN = cm[1][1]
    # FP = cm[0][1]
    P = TP + FN
    N = TN + FP
    TPR = TP/P
    TNR = TN/N
    FPR = FP/N
    FNR = FN/P
    r = TP/P
    p = TP/(TP+FP)
    F1 = 2*(p*r)/(p+r)
    Acc = (TP+TN)/(P+N)
    Err = (FP+FN)/(P+N)
    BACC = (TPR + TNR)/2
    TSS = TP/(TP+FN) - FP/(FP+TN)
    HSS = 2*(TP*TN - FP*FN) / ((TP+FN)*(FN+TN) + (TP+FP)*(FP+TN))
    classifier_performance.append([i, TP, TN, FP, FN, P, N, TPR, TNR, FPR, FNR, r, p, F1, Acc, Err, BACC, TSS, HSS])
```

Performance Metrics for 10-fold Random Forest

Index	TP	TN	FP	FN	P	N	TPR	TNR	FPR	FNR	r	p	F1	Acc	Err	BACC	TSS	HSS
0	115	3506	139	359	474	3645	0.242616	0.961866	0.038134	0.757384	0.242616	0.452756	0.315934	0.879097	0.120903	0.602241	0.204482	0.256207
1	119	3537	130	333	452	3667	0.263274	0.964549	0.035451	0.736726	0.263274	0.477912	0.339515	0.887594	0.112406	0.613912	0.227823	0.283672
2	116	3515	131	357	473	3646	0.245243	0.964070	0.035930	0.754757	0.245243	0.469636	0.322222	0.881525	0.118475	0.604657	0.209313	0.264254
3	103	3556	125	335	438	3681	0.235160	0.966042	0.033958	0.764840	0.235160	0.451754	0.309309	0.888322	0.111678	0.600601	0.201202	0.255073
4	102	3548	121	348	450	3669	0.226667	0.967021	0.032979	0.773333	0.226667	0.457399	0.303120	0.886137	0.113863	0.596844	0.193688	0.248728
5	115	3518	152	334	449	3670	0.256125	0.958583	0.041417	0.743875	0.256125	0.430712	0.321229	0.882010	0.117990	0.607354	0.214708	0.261163
6	121	3511	143	344	465	3654	0.260215	0.960865	0.039135	0.739785	0.260215	0.458333	0.331962	0.881767	0.118233	0.610540	0.221080	0.272476
7	135	3495	129	360	495	3624	0.272727	0.964404	0.035596	0.727273	0.272727	0.511364	0.355731	0.881282	0.118718	0.618566	0.237131	0.296957
8	114	3515	127	362	476	3642	0.239496	0.965129	0.034871	0.760504	0.239496	0.473029	0.317992	0.881253	0.118747	0.602312	0.204625	0.260531
9	103	3513	137	365	468	3650	0.220085	0.962466	0.037534	0.779915	0.220085	0.429167	0.290960	0.878096	0.121904	0.591276	0.182551	0.231769

Average Performance Metrics for Random Forest

Method	Random Forest
TP	114.3
TN	3521.4
FP	133.4
FN	349.7
P	464.0
N	3654.8
TPR	0.246161
TNR	0.963499
FPR	0.036501
FNR	0.753839
r	0.246161
p	0.461206
F1	0.320797
Acc	0.882708
Err	0.117292
BACC	0.60483
TSS	0.20966
HSS	0.263083

KNN

```
kf = KFold(n_splits=10)
knn = KNeighborsClassifier()
classifier_performance = []

for i, (train_indices, test_indices) in enumerate(kf.split(X)):
    X_train, y_train = X.iloc[train_indices], y.iloc[train_indices]
    X_test, y_test = X.iloc[test_indices], y.iloc[test_indices]

    knn.fit(X_train, y_train)

    y_pred = knn.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    TN, FP, FN, TP = cm.ravel()
    # TP = cm[0][0]
    # FN = cm[1][0]
    # TN = cm[1][1]
    # FP = cm[0][1]
    P = TP + FN
    N = TN + FP
    TPR = TP/P
    TNR = TN/N
    FPR = FP/N
    FNR = FN/P
    r = TP/P
    p = TP/(TP+FP)
    F1 = 2*(p*r)/(p+r)
    Acc = (TP+TN)/(P+N)
    Err = (FP+FN)/(P+N)
    BACC = (TPR + TNR)/2
    TSS = TP/(TP+FN) - FP/(FP+TN)
    HSS = 2*(TP*TN - FP*FN) / ((TP+FN)*(FN+TN) + (TP+FP)*(FP+TN))
    classifier_performance.append([i, TP, TN, FP, FN, P, N, TPR, TNR, FPR, FNR, r, p, F1, Acc, Err, BACC, TSS, HSS])
```

Performance Metrics for 10-fold KNN

Index	TP	TN	FP	FN	P	N	TPR	TNR	FPR	FNR	r	p	F1	Acc	Err	BACC	TSS	HSS
0	78	3555	113	373	451	3668	0.172949	0.969193	0.030807	0.827051	0.172949	0.408377	0.242991	0.882010	0.117990	0.571071	0.142142	0.190235
1	101	3560	112	346	447	3672	0.225951	0.969499	0.030501	0.774049	0.225951	0.474178	0.306061	0.888808	0.111192	0.597725	0.195450	0.253792
2	120	3504	128	367	487	3632	0.246407	0.964758	0.035242	0.753593	0.246407	0.483871	0.326531	0.879825	0.120175	0.605582	0.211164	0.268138
3	112	3514	137	356	468	3651	0.239316	0.962476	0.037524	0.760684	0.239316	0.449799	0.312413	0.880311	0.119689	0.600896	0.201792	0.253502
4	108	3540	117	354	462	3657	0.233766	0.968007	0.031993	0.766234	0.233766	0.480000	0.314410	0.885652	0.114348	0.600886	0.201773	0.260047
5	102	3532	117	368	470	3649	0.217021	0.967936	0.032064	0.782979	0.217021	0.465753	0.296081	0.882253	0.117747	0.592479	0.184958	0.241028
6	103	3531	123	362	465	3654	0.221505	0.966338	0.033662	0.778495	0.221505	0.455752	0.298119	0.882253	0.117747	0.593922	0.187844	0.242156
7	96	3542	105	376	472	3647	0.203390	0.971209	0.028791	0.796610	0.203390	0.477612	0.285290	0.883224	0.116776	0.587300	0.174599	0.232775
8	109	3557	115	337	446	3672	0.244395	0.968682	0.031318	0.755605	0.244395	0.486607	0.325373	0.890238	0.109762	0.606538	0.213077	0.272703
9	108	3513	133	364	472	3646	0.228814	0.963522	0.036478	0.771186	0.228814	0.448133	0.302945	0.879310	0.120690	0.596168	0.192335	0.244398

Average Performance Metrics for KNN

Method	KNN
TP	103.7
TN	3534.8
FP	120.0
FN	360.3
P	464.0
N	3654.8
TPR	0.223351
TNR	0.967162
FPR	0.032838
FNR	0.776649
r	0.223351
p	0.463008
F1	0.301021
Acc	0.883388
Err	0.116612
BACC	0.595257
TSS	0.190513
HSS	0.245877

LSTM

```
kf = KFold(n_splits=10)
classifier_performance = []

for i, (train_indices, test_indices) in enumerate(kf.split(X)):
    X_train, y_train = X.iloc[train_indices], y.iloc[train_indices]
    X_test, y_test = X.iloc[test_indices], y.iloc[test_indices]

    model.fit(X_train.to_numpy(), y_train.to_numpy(), batch_size=len(X_train), epochs=10)

    y_pred = model.predict(X_test)
    y_pred[y_pred <= 0] = 0
    y_pred[y_pred > 0] = 1
    y_pred

    cm = confusion_matrix(y_test, y_pred)
    TN, FP, FN, TP = cm.ravel()
    # TP = cm[0][0]
    # FN = cm[1][0]
    # TN = cm[1][1]
    # FP = cm[0][1]
    P = TP + FN
    N = TN + FP
    TPR = TP/P
    TNR = TN/N
    FPR = FP/N
    FNR = FN/P
    r = TP/P
    p = TP/(TP+FP)
    F1 = 2*(p*r)/(p+r)
    Acc = (TP+TN)/(P+N)
    Err = (FP+FN)/(P+N)
    BACC = (TPR + TNR)/2
    TSS = TP/(TP+FN) - FP/(FP+TN)
    HSS = 2*(TP*TN - FP*FN) / ((TP+FN)*(FN+TN) + (TP+FP)*(FP+TN))
    classifier_performance.append([i, TP, TN, FP, FN, P, N, TPR, TNR, FPR, FNR, r, p, F1, Acc, Err, BACC, TSS, HSS])
```

Performance Metrics for 10-fold LSTM

Index	TP	TN	FP	FN	P	N	TPR	TNR	FPR	FNR	r	p	F1	Acc	Err	BACC	TSS	HSS
0	207	2059	1620	233	440	3679	0.470455	0.559663	0.440337	0.529545	0.470455	0.113300	0.182620	0.550134	0.449866	0.515059	0.030117	0.012614
1	205	2086	1600	228	433	3686	0.473441	0.565925	0.434075	0.526559	0.473441	0.113573	0.183199	0.556203	0.443797	0.519683	0.039366	0.016415
2	219	2055	1567	278	497	3622	0.440644	0.567366	0.432634	0.559356	0.440644	0.122620	0.191853	0.552076	0.447924	0.504005	0.008010	0.003780
3	215	2095	1546	263	478	3641	0.449791	0.575391	0.424609	0.550209	0.449791	0.122090	0.192050	0.560816	0.439184	0.512591	0.025182	0.011627
4	209	2126	1551	233	442	3677	0.472851	0.578189	0.421811	0.527149	0.472851	0.118750	0.189827	0.566885	0.433115	0.525520	0.051039	0.022078
5	214	2079	1552	274	488	3631	0.438525	0.572570	0.427430	0.561475	0.438525	0.121178	0.189885	0.556689	0.443311	0.505547	0.011094	0.005200
6	213	2047	1589	270	483	3636	0.440994	0.562981	0.437019	0.559006	0.440994	0.118202	0.186433	0.548677	0.451323	0.501988	0.003975	0.001820
7	202	2050	1595	272	474	3645	0.426160	0.562414	0.437586	0.573840	0.426160	0.112410	0.177895	0.546735	0.453265	0.494287	-0.011425	-0.005160
8	206	2110	1533	269	475	3643	0.433684	0.579193	0.420807	0.566316	0.433684	0.118459	0.186089	0.562409	0.437591	0.506439	0.012877	0.005970
9	202	2088	1600	228	430	3688	0.469767	0.566161	0.433839	0.530233	0.469767	0.112098	0.181004	0.556095	0.443905	0.517964	0.035928	0.014912

Performance Metrics for LSTM

Method	LSTM
TP	209.2
TN	2079.5
FP	1575.3
FN	254.8
P	464.0
N	3654.8
TPR	0.451631
TNR	0.568985
FPR	0.431015
FNR	0.548369
r	0.451631
p	0.117268
F1	0.186085
Acc	0.555672
Err	0.444328
BACC	0.510308
TSS	0.020616
HSS	0.008926

Comparing the Three Models

Method	Random Forest
TP	114.3
TN	3521.4
FP	133.4
FN	349.7
P	464.0
N	3654.8
TPR	0.246161
TNR	0.963499
FPR	0.036501
FNR	0.753839
r	0.246161
p	0.461206
F1	0.320797
Acc	0.882708
Err	0.117292
BACC	0.60483
TSS	0.20966
HSS	0.263083

Method	KNN
TP	103.7
TN	3534.8
FP	120.0
FN	360.3
P	464.0
N	3654.8
TPR	0.223351
TNR	0.967162
FPR	0.032838
FNR	0.776649
r	0.223351
p	0.463008
F1	0.301021
Acc	0.883388
Err	0.116612
BACC	0.595257
TSS	0.190513
HSS	0.245877

Method	LSTM
TP	209.2
TN	2079.5
FP	1575.3
FN	254.8
P	464.0
N	3654.8
TPR	0.451631
TNR	0.568985
FPR	0.431015
FNR	0.548369
r	0.451631
p	0.117268
F1	0.186085
Acc	0.555672
Err	0.444328
BACC	0.510308
TSS	0.020616
HSS	0.008926

Looking at accuracy, the models ranked from strongest to weakest are:

1. KNN (Acc: 0.883388)
2. Random Forest (Acc: 0.882708)
3. LSTM (Acc: 0.555672)

LSTM has by far the lowest metrics, compared to KNN and Random Forest. Random Forest and KNN have very similar metric values. Random Forest has a higher TPR and TNR but a lower p than KNN.