Rebecca Wood

Artificial Intelligence Final – Part 2

Objective

The objective of my model creation is to predict whether or not a particular client would access a bank term deposit for a Portuguese banking institution based on phone call information only for clients who have successfully given their information. Data collected about these calls includes information about the client's age, employment status, education status, marital status, banking information, and various information about the bank.

Final Classification Code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
import numpy as np
import time

# Data Clean-Up
data = pd.read_csv('bank-additional-full.csv',
                   delimiter = ';')
print(data.shape)

data = data[data.job != 'unknown']
data = data[data.marital != 'unknown']
data = data[data.education != 'unknown']
data = data[data.default != 'unknown']
data = data[data.housing != 'unknown']
data = data[data.loan != 'unknown']
data = data[data.contact != 'unknown']

data['variation'] = data['emp.var.rate']
data['price'] = data['cons.price.idx']
data['confidence'] = data['cons.conf.idx']
data['employees'] = data['nr.employed']

def to_int(data, new_col, current_col):
```

```python
    data[new_col] = data[current_col].apply(lambda x: 0 if x=='no' else 1)
    return data[new_col].value_counts()

to_int(data, ‘def_int’, ‘default’)
to_int(data, ‘house_int’, 'housing')
to_int(data, 'loan_int', 'loan')
to_int(data, 'response', 'y')

date_fix = [data]
for column in date_fix:
    column.loc[column['month'] == 'jan', 'month_int'] = 1
    column.loc[column['month'] == 'feb', 'month_int'] = 2
    column.loc[column['month'] == 'mar', 'month_int'] = 3
    column.loc[column['month'] == 'apr', 'month_int'] = 4
    column.loc[column['month'] == 'may', 'month_int'] = 5
    column.loc[column['month'] == 'jun', 'month_int'] = 6
    column.loc[column['month'] == 'jul', 'month_int'] = 7
    column.loc[column['month'] == 'aug', 'month_int'] = 8
    column.loc[column['month'] == 'sep', 'month_int'] = 9
    column.loc[column['month'] == 'oct', 'month_int'] = 10
    column.loc[column['month'] == 'nov', 'month_int'] = 11
    column.loc[column['month'] == 'dec', 'month_int'] = 12
    column.loc[column['day_of_week'] == 'mon', 'day_int'] = 1
    column.loc[column['day_of_week'] == 'tue', 'day_int'] = 2
    column.loc[column['day_of_week'] == 'wed', 'day_int'] = 3
    column.loc[column['day_of_week'] == 'thu', 'day_int'] = 4
    column.loc[column['day_of_week'] == 'fri', 'day_int'] = 5
    column.loc[column['job'] == 'admin.', 'emp'] = 10
    column.loc[column['job'] == 'self-employed', 'emp'] = 4
    column.loc[column['job'] == 'blue-collar', 'emp'] = 9
    column.loc[column['job'] == 'entrepreneur', 'emp'] = 8
    column.loc[column['job'] == 'housemaid', 'emp'] = 7
    column.loc[column['job'] == 'management', 'emp'] = 6
    column.loc[column['job'] == 'retired', 'emp'] = 3
    column.loc[column['job'] == 'services', 'emp'] = 5
    column.loc[column['job'] == 'student', 'emp'] = 2
    column.loc[column['job'] == 'technician', 'emp'] = 11
    column.loc[column['job'] == 'unemployed', 'emp'] = 1
    column.loc[column['marital'] == 'married', 'mar'] = 2
    column.loc[column['marital'] == 'divorced', 'mar'] = 3
    column.loc[column['marital'] == 'single', 'mar'] = 1
    column.loc[column['education'] == 'basic.4y', 'edu'] = 1
    column.loc[column['education'] == 'basic.6y', 'edu'] = 2
    column.loc[column['education'] == 'basic.9y', 'edu'] = 3
    column.loc[column['education'] == 'high.school', 'edu'] = 4
    column.loc[column['education'] == 'professional.course', 'edu'] = 5
    column.loc[column['education'] == 'university.degree', 'edu'] = 6
    column.loc[column['education'] == 'illiterate', 'edu'] = 7

data['month_int'] = data['month_int'].astype(np.int64)
```

```
data['day_int'] = data['day_int'].astype(np.int64)
data['emp'] = data['emp'].astype(np.int64)
data['mar'] = data['mar'].astype(np.int64)
data['edu'] = data['edu'].astype(np.int64)

data['prev'] = data['pdays'].apply(lambda x: 1 if x != 999 else (1 if x == 0
      else 0))
data['cont'] = data['contact'].apply(lambda x: 1 if x == 'cellular' else 2)

data['outcome'] = data['poutcome'].apply(lambda x: 0 if x == 'failure' else
      (1 if x == 'success' else np.NaN))
data = data.dropna()
data['outcome'] = data['outcome'].astype(np.int64)

data.drop(['job', 'marital', 'education', 'contact', 'emp.var.rate',
      'cons.price.idx', 'cons.conf.idx', 'nr.employed', 'poutcome',
      'default', 'housing', 'loan', 'y', 'duration', 'day_of_week', 'month',
      'pdays'], axis = 1, inplace = True)

# Create test/train sets
y = data['outcome']
x = data.iloc[:,0:19]

x_minmax = preprocessing.normalize(x, norm = 'l2')
x = preprocessing.scale(x_minmax)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = .2)

'''
A reference from Kaggle was used to help effectively compare the various
models -
https://www.kaggle.com/janiobachmann/bank-classifying-term-deposit-
subscriptions
'''
class_dict = {'Logistic Regression': LogisticRegression(),
              'KNN': KNeighborsClassifier(),
              'Linear SVM': SVC(),
              'Gradient Boosting': GradientBoostingClassifier(),
              'Decision Tree': tree.DecisionTreeClassifier(),
              'Random Forest': RandomForestClassifier(n_estimators = 100),
              'Neural Net': MLPClassifier(alpha = 0.1),
              'Naive Bayes': GaussianNB()}

no_class = len(class_dict.keys())

def batch_class(c_train, y_train, verbose = True):
    df_result = pd.DataFrame(data = np.zeros(shape = (no_class, 8)), columns
            = ['classifier', 'trainScore', 'testScore', 'truePos', 'trueNeg',
            'falsePos', 'falseNeg', 'time'])
    count = 0
```

```
    for key, classifier in class_dict.items():
        t_start = time.clock()
        classifier.fit(x_train, y_train)
        y_hat = classifier.predict(x_test)
        t_end = time.clock()
        t_diff = t_end - t_start
        train_score = classifier.score(x_train, y_train)
        tn, fp, fn, tp = confusion_matrix(y_test, y_hat).ravel()
        df_result.loc[count, 'classifier'] = key
        df_result.loc[count, 'trainScore'] = train_score*100
        df_result.loc[count, 'time'] = t_diff
        df_result.loc[count, 'testScore'] = accuracy_score(y_hat, y_test)*100
        df_result.loc[count, 'truePos'] = float(tp)/(tp + fn)*100
        df_result.loc[count, 'trueNeg'] = float(tn)/(tn+fp)*100
        df_result.loc[count, 'falsePos'] = float(fp)/(fp+tn)*100
        df_result.loc[count, 'falseNeg'] = float(fn)/(tp+fn)*100
        if verbose:
            print( ‘trained {c} in {f:.2f} s’.format(c = key, f = t_diff))
        count += 1
    return df_result

df_result = batch_class(x_train, y_train)
df_final = df_result.sort_values(by = 'testScore', ascending = False).copy()

writer = pd.ExcelWriter('finaltable.xlsx')
df_final.to_excel(writer, 'Sheet1')
writer.save()

# Print table of Model Comparisons
print('_______________________________________')
print(df_final)
print('** Score values given as percents **')

# Print Best Model Results
print('_______________________________________')
print(‘Best Model: \n’)
print(np.transpose(df_final.head(1)))

# Print Neural Network Model Results
print('_________________________________________')
print('Classification Neural Network: \n')
print(df_final.loc[1,])
```

Code Explanation

Packages

Pandas and Numpy were imported to help with data frame management and missing numeric values. The time package was used to calculate the model computation time.

Model selection, metrics, preprocessing, linear model, support vector machine, neighbors, tree, neural network, ensemble, and Naïve-Bayes were imported from the SKLearn package to train, test, and validate the final model.

Data Clean-Up

A seed was not set for this assignment to allow for any variations between runs and to determine how well the neural network would preform against the machine learning techniques with each run. After importing the CSV file, any rows with 'unknown' values were removed. In addition, column names with periods were remained to enable the use of different slicing methods in Python. Within the original data set, any numeric values were a string type. A new column was created for each of the four columns with numeric values so those values could be converted to integers. As discussed in the proposal for this project, six columns contained groups of classifiers that were converted to integer values for convenience. The *'pdays'* column was converted into a new metric; if clients were contacted prior to the current contact they were given a value of 1, if they were not contacted they were given a value of 0. Finally, in the 'poutcomes' column, there were some missing values which were given the 'NaN' values to later be dropped. Finally, the original columns that were recreated in the data clean-up process were deleted to maintain the 20 predictors.

Model Development and Output

The 20 predictors were assigned to the *x*-variable while the *'outcome'* column (previously the *'poutcome'* column) is assigned to the *y*-variable containing about 4600 observations. From that the x values were normalized using the l2-norm and then scaled. This simply centers the data with a standard deviation of 1 and mean of 0. Finally, the train-test-split function was used to give a test size of 20%. There was no significance to choosing an 80-20 split, other than that this is common practice. Using a reference from Kaggle, a dictionary was created to easily filter through all of the machine learning and neural network functions. Logistic regression, K-nearest neighbors, linear support vector machines, gradient boosting, decision trees, random forest, MLP classifier, and Naïve-Bayes were the chosen methods for model development. These methods are efficient, easy to compute, and relatively accurate in terms of classification. For the neural network, all parameters are defaults except an alpha value was set at 0.1. The default settings are to use the rectified linear unit function was used as the activator with 100 hidden layers, a stochastic gradient-based method (adam) as a solver, a learning rate of 0.0001, and 200 iterations. The momentum is 0.90; however, because stochastic gradient descent was not the solver, this parameter was unused. Each model was trained to fit the training set where the training accuracy score was calculated. The model was then used to predict responses for the test set, where the testing accuracy score was also calculated. Further metrics, such as the true positive, false positive, true negative, false negative rates were also calculated and saved as percentages. The final output was a pandas data frame with each of the values appended including the computation time for each model and sorted by the best testing accuracy score. In the console the best model data was printed (model with the highest testing accuracy score), as well as the neural network data. This table was saved as a CSV and can be found with in the results section of this write-up.

Results

```
            classifier  trainScore  testScore     truePos    trueNeg  \
4          Naive Bayes   97.285676  97.959184  100.000000  97.304965
5           Linear SVM   97.473797  97.959184  100.000000  97.304965
7        Random Forest  100.000000  97.959184   98.672566  97.730496
6    Gradient Boosting   98.763773  97.851772   98.672566  97.588652
1           Neural Net   98.091911  97.744361   97.787611  97.730496
2  Logistic Regression   97.097554  97.744361   99.115044  97.304965
3        Decision Tree  100.000000  97.529538   95.132743  98.297872
0                  KNN   96.801935  95.918367   92.477876  97.021277

   falsePos  falseNeg      time
4  2.695035  0.000000  0.002942
5  2.695035  0.000000  0.161360
7  2.269504  1.327434  0.594881
6  2.411348  1.327434  0.353408
1  2.269504  2.212389  1.081149
2  2.695035  0.884956  0.022319
3  1.702128  4.867257  0.014357
0  2.978723  7.522124  0.122917
** Score Values given as percents **
_______________________________________
Best Model:

                      4
classifier  Naive Bayes
trainScore      97.2857
testScore       97.9592
truePos             100
trueNeg          97.305
falsePos        2.69504
falseNeg              0
time         0.00294181
_________________________________________
Classification Neural Network:

classifier    Neural Net
trainScore       98.0919
testScore        97.7444
truePos          97.7876
trueNeg          97.7305
falsePos          2.2695
falseNeg         2.21239
time             1.08115
Name: 1, dtype: object

In [3]:
```

The output into the console is given to the right. First, the final aggregated table was printed showing the models, their training and testing accuracies, error ratings and computation time. The table was also printed in an Excel spreadsheet (shown below). The machine learning techniques appeared to preform better than the neural network on average. The neural network also took the most amount of computation time (about 1 second) and may prove inefficient when working with a larger data set. The most accurate model was the Naïve-Bayes model; however, the training accuracy was below average with a relatively high false postive rate. Sacrificing the type 1 errors for a quick computation time, no type 2 errors is reasonable considering all of the models preformed almost the same. Due to the seed not being set it should be noted that the top preforming will change; however, after running the model about 15 times it was noted that the neural network placed no higher than third for each run.

	classifier	trainScore	testScore	truePos	trueNeg	falsePos	falseNeg	time
4	Naive Bayes	97.28567589	97.95918367	100.00000000	97.30496454	2.69503546	0.00000000	0.00294181
5	Linear SVM	97.47379737	97.95918367	100.00000000	97.30496454	2.69503546	0.00000000	0.16135972
7	Random Forest	100.00000000	97.95918367	98.67256637	97.73049645	2.26950355	1.32743363	0.59488146
6	Gradient Boosting	98.76377318	97.85177229	98.67256637	97.58865248	2.41134752	1.32743363	0.35340841
1	Neural Net	98.09191078	97.74436090	97.78761062	97.73049645	2.26950355	2.21238938	1.08114937
2	Logistic Regression	97.09755442	97.74436090	99.11504425	97.30496454	2.69503546	0.88495575	0.02231939
3	Decision Tree	100.00000000	97.52953813	95.13274336	98.29787234	1.70212766	4.86725664	0.01435714
0	KNN	96.80193496	95.91836735	92.47787611	97.02127660	2.97872340	7.52212389	0.12291708