ASSIGNMENT: DESIGN AND APPLICATION OF A MACHINE LEARNING SYSTEM FOR A PRACTICAL PROBLEM

A REPORT

submitted as part of the assignment

in

CE802 Machine Learning and Data Mining

by

ISHWAR VENUGOPAL

(1906084)

School of Computer Science and Electronic Engineering
University of Essex

January 2020

CONTENTS

PILOT-STUDY PROPOSAL

(Word count:725)

1. PILOT-STUDY PROPOSAL

Fake news and misleading information have been a centre of debate and discussion in recent times. The increasing influence of social media on the public have been able to propagate ideas ranging from marketing strategies to election propaganda. This is a serious issue and the need to identify misleading false information on the social media is essential. The task at hand is to be able to predict if a given account is likely to post false news on the social media based on the details available regarding that particular account.

As the problem statement clearly outlines, this is a matter of predicting a binary outcome given a set of features for a user account. The output shall give us the information about the possibility of that account to post fake news on social media. Framing the question as, 'Will the given user be posting a possibly fake news in the future on social media?'; the answer could be 'Yes' or 'No'. Hence, we can categorize this scenario as a classification problem in a machine learning perspective.

In order to identify the characteristics of a given user account, we need to have a list of useful feature parameters and their corresponding values for each account. This should possibly include the following features:

- How long has the account been active on that social media platform
- The age and gender of the person
- The profession of the user
- Whether the person is affiliated to any political party
- The Education status of the person
- Any known link of the user to newspapers or other platforms which have been previously identified to have spread fake or misleading news

- In order to identify dummy accounts, we would also need to have the number of posts that the user interacts with every day (likes, comments, shares etc.)
- The number of posts that the user has interacted with, which were directly linked to propagating false news on the social media platform

The features should include but not necessarily be limited to the ones compiled above. Often, user accounts which have a known link to any previously published false news or posts are more likely to be involved in propagating misleading information in the future.

Once we have the features at hand, we need to employ the appropriate machine learning classification technique to approach this problem. Some of the techniques we could try on this data set are Decision Trees, Naive Bayes Classifiers, k-Nearest Neighbour classifiers and Support Vector Machines. However, Decision trees are known to be greedy algorithms compared to the others. And the possibility of the tree getting more and more complex with increasing number of features and data is also is an issue worth keeping in mind. On the other hand Bayesian models are simpler and use much lesser data than other models. K-nearest neighbour method and Support Vector Machines are much more sophisticated models which can be used for classification problems and are observed to give better results in many cases. Once each of these methods have been tried on the data set, the most suitable model can be chosen by comparing the accuracy measure, Kappa scores and the confusion matrices. Intuitively, higher the accuracy of the model, better the model is in itself. It has been widely accepted that the Kappa score can be interpreted as follows: values which are less than or equal to 0 indicates that there is no agreement between the predicted outcome and the true outcome, 0.01–0.20 is seen to illustrate none to slight agreement, 0.21–0.40 is a fair score for agreement , 0.41– 0.60 is a moderate score, 0.61–0.80 is accepted to be a substantial score and 0.81–1.00 is agreed upon to be an almost perfect agreement. Confusion matrices help give us an idea of the number of True positives, true negatives, false positives and false negatives that emerge out of the model. These metrics and scores can be analyzed to obtain the final model to be implemented.

Even though machine learning techniques can to some extent predict the spreading of false news, it is not possible to completely eradicate the process using automated systems. Deploying a separate team to work on the results obtained from a machine learning model and have human inputs and suggestions to the results is something the company must

consider seriously as the next steps.

COMPARATIVE STUDY

(Word count:1023)

2. COMPARATIVE STUDY

2.1 Data Pre-Processing

With the data provided by the company each account has 20 features associated to it. There are 500 such accounts for which the data has been provided. Each of the account has a corresponding column having binary values (either 'True' or 'False'), which denotes if it is observed to have spread false or misleading news/information. We have used this data set for training all of our models described hereafter.

The first step towards processing the data was to address the null values that occurred in the column containing values for the 20th feature. This has been resolved by replacing all the NaN values with zero. Once we try different models on this data, we will revisit this aspect to understand how other processing methods would have influenced the result. Also, all TRUE values were replaced by 1 and FALSE values were replaced by 0.

After the data has been processed to remove the null values, we then use the *train_test_split* [9] function available within the *sklearn.preprocessing* package. Here we have split the data in such a way that the test data would then have 25 % of the data. We also use the *StandardScaler* [7] function to rescale the values before applying the training process. This rescales the input values as per the given equation:

$$x^{|} = \frac{x - \mu}{\sigma} \tag{2.1}$$

where μ denotes the mean and σ denotes the standard deviation. Once we have completed all these steps, we are ready to start implementing and testing different models on this data.

2.2 Machine Learning Models

2.2.1 Decision Trees

We first begin by implementing the decision tree classifier [4]. The model we have implemented in the code [refer Appendix] identifies the relevance of the features based on the information gain. The parameter 'criterion' in the *DecisionTreeClassifier* can be set to *entropy* to be able to use the information gain as the deciding measure. The model is trained over the pre-processed data. And the following confusion matrix [3] was generated:

	Predicted = NO	Predicted = YES
Actual = NO	43	27
Actual = YES	31	24

Table 2.1: Confusion matrix generated from the decision tree classifier

2.2.2 K-Nearest Neighbour Classifier

In the K-nearest neighbour classifier [6] that we have implemented, the distance between two feature points is calculated with the Minkowski distance. The other possible distance metrics we could have used was the Euclidian distance, Manhattan distance etc [5]. The initial number of nearest neighbours to use by default was set as 6 for the first trial. After training the data, we got the following confusion matrix [3]:

	Predicted = NO	Predicted = YES
Actual = NO	62	8
Actual = YES	35	20

Table 2.2: Confusion matrix generated from the K-Nearest neighbour classifier

To identify the ideal distance metrics, different models were run by the keeping the initial number of default nearest neighbours as the same. The results were summarised as follows:

DISTANCE METRICS	ACCURACY
Minkowski	0.656
Euclidean	0.656
Chebyshev	0.616
Manhattan	0.624

Table 2.3: Comparing different distance metrics for K-Nearest neighbour classifier

It can be seen that the highest accuracy was given by the models that used Minkowski or Euclidian distance metrics. Similarly in order to find the optimum number of initial nearest neighbours, different ensembles were run by varying just the number of initial nearest neighbours (from 2 to 10).

NUMBER OF NEAREST NEIGHBOURS	ACCURACY
2	0.64
3	0.68
4	0.64
5	0.624
6	0.656
7	0.664
8	0.608
9	0.608
10	0.624

Table 2.4: Comparing different distance metrics for K-Nearest neighbour classifier

Based on the data given above, it was concluded that the initial number of nearest neighbours should neither be too small nor be too high. Keep the risks of underfitting and overfitting in mind, the initial number of nearest neighbours was chosen to be 6.

2.2.3 Support Vector Machines

Support Vector machine was another type of classifier model that we implemented on the pre-processed data [8]. We used a 'rbf' kernel for this problem [refer 2.6]. After training

the SVM model with the given data, the confusion matrix [3] generated was as follows:

	Predicted = NO	Predicted = YES
Actual = NO	54	16
Actual = YES	30	25

Table 2.5: Confusion matrix generated from the SVM classifier

The model was tested by using different types of kernels. The different ensembles and their corresponding accuracies are summarised in the table below:

KERNEL	ACCURACY
linear	0.6
poly	0.608
rbf	0.632
sigmoid	0.52

Table 2.6: Comparing different kernels for the SVM classifier

2.3 Comparison of the models

We have already had a look at the confusion matrices generated from each of the models that we have generated. To get a better understanding of the models, we compare the accuracy scores [1] as the Cohen Kappa scores [2]. The Kappa scores gives the extent to which two annotators agree to each other. In our case, we have used the Kappa score to determine the extent to which the predicted values agree to the true output values. The accuracy scores and the Kappa scores for each of the model has been summarized as in Table 2.7.

We can thus conclude that the K-Nearest neighbour classifier gives the best results for the data presented to us. The decision tree might have given poor accuracy as expected because the more number of features in the training data might have complicated the trees that were generated. SVM and the K-NN model gave similar performance measures. K-NN seems to be a better suited model for this problem because it can clearly (to a large extent) classify the behaviour of the user accounts based on similar features they share.

MODEL	ACCURACY	KAPPA SCORE
Decision Tree Classifier	0.536	0.05104712041884829
K-NN Classifier	0.656	0.26319396847155585
SVM Classifier	0.632	0.17871222076215498

Table 2.7: Comparison of the different classifier models

2.4 Comparison of Data Processing Methods

To deal with the null values that appeared in the training data, we could have used any of the following methods:

- Method 1: Remove the rows containing the NULL element
- Method 2: Replace the null values with zero
- Method 3: Replace the null values with the mode of the columns
- Method 4: Replace the null values with the mean of the columns

We carried out each of the data pre-processing methods discussed above and have summarised the results from a k-Nearest neighbour classifier on that data, in a table as follows:

METHOD	ACCURACY
Method 1	0.6219512195121951
Method 2	0.656
Method 3	0.664
Method 4	0.648

Table 2.8: Comparing different data pre-processing methods for NULL values

It can be observed from Table 2.8 that replacing the NaN values with the mode value of that particular column gave the best results compared to all other methods.

As a concluding statement, we can thus infer that a K-nearest neighbour classification model (with parameters: number of initial nearest neighbours = 6, distance metrics =

Minkowski distance) trained on a data set which is processed by replacing all the NaN values with the mode value of that particular column, can give the best predictions for an unseen test data.

BIBLIOGRAPHY

[1] Accuracy score. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html.

[2] Cohen kappa score. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.cohen_kappa_score.html.

[3] Confusion matrix. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html.

[4] Decision tree classifier. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

[5] Distance metric. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.DistanceMetric.html.

[6] K neighbors classifier. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.

[7] Standard scaler. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

[8] Svc. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

[9] Train test split. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

Appendices

Decision_Tree_Assignment

January 10, 2020

In [2]:
```
# Importing the dataset
dataset = pd.read_csv('CE802_Ass_2019_Data.csv')
dataset.head()
```

Out[2]:

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	...	F12	F13	F14	\
0	0	0	16	2.02	0.52	-2.35	-1.98	-0.70	85	6	...	-0.07	1.08	15	
1	0	0	86	-0.90	2.75	0.14	0.83	-0.06	107	1	...	0.17	1.06	-8	
2	1	1	165	0.73	1.05	0.10	2.57	-1.65	41	5	...	0.04	0.42	-6	
3	1	1	191	-1.50	0.79	0.33	1.24	1.35	17	2	...	1.74	1.74	15	
4	1	1	13	0.25	-1.19	-0.90	2.67	0.22	12	8	...	-0.39	1.25	25	

	F15	F16	F17	F18	F19	F20	Class
0	-0.63	-3.49	-1.68	0.02	15.3	0.00	1
1	-1.21	0.34	0.36	0.61	10.1	0.00	1
2	-0.46	-0.62	1.67	2.60	11.0	1.55	0
3	0.47	0.63	0.08	0.19	6.3	0.95	0
4	-0.09	-2.41	-0.53	-0.77	10.5	0.00	1

[5 rows x 21 columns]

In [23]:
```
X = dataset.iloc[:, 0:20]
#X.head()
y = dataset.iloc[:, [20]]
#y.head()
```

In [24]:
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,random_state
```

In [31]:
```
from sklearn.preprocessing import
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [33]:
```
# Fitting Decision Tree Classification to the Training set
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy',
                                    random_state = 0)
classifier.fit(X_train, y_train)
```

```
Out[33]: DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
                                max_features=None, max_leaf_nodes=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, presort=False,
                                random_state=0, splitter='best')

In [34]: # Predicting the Test set results
         y_pred = classifier.predict(X_test)

In [35]: # Making the Confusion Matrix
         from sklearn.metrics import confusion_matrix
         cm = confusion_matrix(y_test, y_pred)
         cm

Out[35]: array([[43, 27],
                [31, 24]], dtype=int64)

In [36]: from sklearn.metrics import accuracy_score
         score = accuracy_score(y_test, y_pred)
         score

Out[36]: 0.536

In [38]: from sklearn.metrics import cohen_kappa_score
         cohen_kappa_score(y_test, y_pred)

Out[38]: 0.05104712041884829
```

K_nn Assignment

January 10, 2020

In [156]:
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [157]:
```
# Importing the dataset
dataset = pd.read_csv('CE802_Ass_2019_Data.csv')
dataset.head()
```

Out[157]:

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	...	F12	F13	F14	\
0	0	0	16	2.02	0.52	-2.35	-1.98	-0.70	85	6	...	-0.07	1.08	15	
1	0	0	86	-0.90	2.75	0.14	0.83	-0.06	107	1	...	0.17	1.06	-8	
2	1	1	165	0.73	1.05	0.10	2.57	-1.65	41	5	...	0.04	0.42	-6	
3	1	1	191	-1.50	0.79	0.33	1.24	1.35	17	2	...	1.74	1.74	15	
4	1	1	13	0.25	-1.19	-0.90	2.67	0.22	12	8	...	-0.39	1.25	25	

	F15	F16	F17	F18	F19	F20	Class
0	-0.63	-3.49	-1.68	0.02	15.3	-0.13	1
1	-1.21	0.34	0.36	0.61	10.1	-0.13	1
2	-0.46	-0.62	1.67	2.60	11.0	1.55	0
3	0.47	0.63	0.08	0.19	6.3	0.95	0
4	-0.09	-2.41	-0.53	-0.77	10.5	-0.13	1

[5 rows x 21 columns]

In [158]:
```
X = dataset.iloc[:, 0:20]
#X.head()
y = dataset.iloc[:, [20]]
#y.head()
```

In [159]:
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,random_stat
```

In [160]:
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
In [161]: # Fitting classifier to the Training set
          from sklearn.neighbors import KNeighborsClassifier
          classifier = KNeighborsClassifier(n_neighbors=6, metric='minkowski', p=2)
          classifier.fit(X_train,y_train)

C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: DataConversionWarning: A col
  after removing the cwd from sys.path.


Out[161]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                               metric_params=None, n_jobs=None, n_neighbors=6, p=2,
                               weights='uniform')

In [162]: # Predicting the Test set results
          y_pred = classifier.predict(X_test)

In [163]: # Making the Confusion Matrix
          from sklearn.metrics import confusion_matrix
          cm = confusion_matrix(y_test, y_pred)
          cm

Out[163]: array([[62,  8],
                 [34, 21]], dtype=int64)

In [164]: from sklearn.metrics import accuracy_score
          score = accuracy_score(y_test, y_pred)
          score

Out[164]: 0.664

In [165]: from sklearn.metrics import cohen_kappa_score
          cohen_kappa_score(y_test, y_pred)

Out[165]: 0.2818057455540356
```

0.1 Predicting

```
In [166]: data_test_set = pd.read_csv('CE802_Ass_2019_Test.csv')
          data_test_set.head()

Out[166]:    F1  F2   F3    F4    F5    F6    F7    F8  F9  F10  ...   F12   F13  F14  \
          0   0   0  155  1.23 -0.75 -0.22  2.07 -0.19  18    6  ...  0.82  0.07  -26
          1   0   0  127 -0.30  0.75 -1.34  0.17  0.39  67    8  ... -0.77  1.15   19
          2   1   1   22  1.06 -1.05  0.83  0.62 -0.63  15   23  ... -0.44  1.31   -5
          3   1   1   29 -1.01  1.06  0.96  2.19 -2.07   8  -26  ... -0.76  1.16   -1
          4   0   0   78  0.10 -1.07  2.20  0.99  1.84  83   29  ...  1.00  0.86   -5

              F15   F16   F17   F18   F19   F20  Class
          0 -0.03  0.59 -0.32 -1.55   6.4  1.51      0
          1  1.93  1.53 -0.13  0.79   3.6 -0.73      0
```

```
2  0.38  0.24 -1.46  0.52   3.7  1.49      0
3 -0.10  0.15  1.29  0.79  13.1  2.25      0
4 -1.02 -1.02 -0.43  0.23  10.4 -2.91      0

[5 rows x 21 columns]
```

In [167]:
```
x1=data_test_set.iloc[:,0:20]
#x1.head()
y1=data_test_set.iloc[:,[20]]
#y1.head()
X_test = sc.transform(x1)
```

In [168]:
```
y_pred = classifier.predict(X_test)
df_pred=pd.DataFrame(y_pred)
df_pred.head(10)
```

Out[168]:

	0
0	0
1	1
2	0
3	0
4	0
5	0
6	0
7	0
8	1
9	0

In [170]:
```
df_pred.to_csv('predicted_final.csv')
```

SVM_Assignment

January 10, 2020

In [23]:
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [24]:
```
# Importing the dataset
dataset = pd.read_csv('CE802_Ass_2019_Data.csv')
dataset.head()
```

Out[24]:

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	...	F12	F13	F14	\
0	0	0	16	2.02	0.52	-2.35	-1.98	-0.70	85	6	...	-0.07	1.08	15	
1	0	0	86	-0.90	2.75	0.14	0.83	-0.06	107	1	...	0.17	1.06	-8	
2	1	1	165	0.73	1.05	0.10	2.57	-1.65	41	5	...	0.04	0.42	-6	
3	1	1	191	-1.50	0.79	0.33	1.24	1.35	17	2	...	1.74	1.74	15	
4	1	1	13	0.25	-1.19	-0.90	2.67	0.22	12	8	...	-0.39	1.25	25	

	F15	F16	F17	F18	F19	F20	Class
0	-0.63	-3.49	-1.68	0.02	15.3	-0.13	1
1	-1.21	0.34	0.36	0.61	10.1	-0.13	1
2	-0.46	-0.62	1.67	2.60	11.0	1.55	0
3	0.47	0.63	0.08	0.19	6.3	0.95	0
4	-0.09	-2.41	-0.53	-0.77	10.5	-0.13	1

[5 rows x 21 columns]

In [25]:
```
X = dataset.iloc[:, 0:20]
#X.head()
y = dataset.iloc[:, [20]]
#y.head()
```

In [26]:
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,random_state
```

In [27]:
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
In [28]: # Fitting classifier to the Training set
         from sklearn.svm import SVC
         classifier = SVC(kernel='rbf', random_state=0)
         classifier.fit(X_train, y_train)
         # Predicting the Test set results
         y_pred = classifier.predict(X_test)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:724: DataConversionWarnin
  y = column_or_1d(y, warn=True)


In [29]: # Making the Confusion Matrix
         from sklearn.metrics import confusion_matrix
         cm = confusion_matrix(y_test, y_pred)
         cm

Out[29]: array([[54, 16],
                [31, 24]], dtype=int64)

In [30]: from sklearn.metrics import accuracy_score
         score = accuracy_score(y_test, y_pred)
         score

Out[30]: 0.624

In [31]: from sklearn.metrics import cohen_kappa_score
         cohen_kappa_score(y_test, y_pred)

Out[31]: 0.21404682274247488
```