TV News Channel Commercial Detection

Using Machine Learning Models

Ehab Mohamed
Lewis University
Illinois, United States

***Abstract*—The project aims to automatically detect commercials in TV news videos. Four supervised machine learning models are examined to predict whether a TV news video shot is a commercial or not. The four algorithms are Logistic Regression, Neural Networks, Random Forests and Gradient Boosting Classifier. A set of news video shots is used for training and testing the machine learning models. The performance results of the Neural Network model show the highest prediction accuracy of 93.13%.**

Keywords—tv news; prediction; machine learning; logistic regression; neural networks; random forests; gradient boosting classifier; bagging; boosting; ensemble.

I. OBJECTIVES

The project aims to develop a classification model which can automatically identify commercials, that occupying almost 40-60% of total air time, in TV news videos. The produced model may find a lot of applications in the domain of television broadcast analysis and monitoring.

II. DATASET DESCRIPTION

On particular news channel, TV news commercials are combinations of video shots uniquely characterized by audio-visual presentation. The used dataset consists of features extracted from 150 Hours of broadcast news videos from 5 different news channels (i.e., 3 Indian and 2 International News channels). The five news channels presented in the dataset are CNNIBN, NDTV 24X7, TIMESNOW, BBC and CNN.

The used dataset represents a set of video shots features extracted from Broadcast News videos from those channels. From each video shot, 7 audio features (i.e., Short term energy, zero crossing rate, Spectral Centroid, spectral Flux, spectral Roll off frequency, fundamental frequency and MFCC Bag of Audio Words) and 5 visual features (i.e., Video shot length, Screen Text Distribution, Motion Distribution, Frame Difference Distribution, Edge Change Ratio) are extracted.

The dataset is hosted and can be found on the Machine Learning Repository for University of California, Irvine at: http://archive.ics.uci.edu/ml/datasets/TV+News+Channel+Commercial+Detection+Dataset

The dataset consists of 129,685 instances, where approximately 63% of the instances are commercial instances, and 4,125 features (i.e., attributes). Each instance in the dataset is classified either as “Commercials” or “Non-Commercials”.

A. Pre-Processing

The first stage in the project is the pre-processing stage which aims to prepare the data in a way that can be smoothly processed by the prediction models. This stage consists of four different tasks which are merging data, splitting data into training and testing datasets, standardizing data, and selecting features.

1) Merging Data: aims to merge data generated from the five news channels into a single dataset.

2) Splitting Data into Training and Testing Datasets: aims to generate two different datasets from the original dataset, one for training and the other for testing. The training dataset is 80% (i.e., 103,748 instances) of the original dataset, while the testing dataset is 20% (i.e., 25,937 instances).

3) Standardizing Data: aims to standardize the data by subtracting the mean of the corresponding attribute from each data value and dividing it by the standard deviation.

4) Selecting Features Using Principle Component Analysis (PCA): aims to reduce the number of features by selecting the features which explain 80% of the variance. The task utilizes the PCA to identify the required features and to transform the dataset accordingly. After performing this task, the number of features has been reduced from 4,125 to only 76 features.

III. PREDICTION PROCESS

Four machine learning models are used to perform the prediction task, which are Logistic Regression, Neural Network, Random Forests, and Gradient Boosting Classifier.

A. Logistic Regression (LR)

LR is one of the simplest models which requires no assumptions about the underlying likelihood distribution, however, the classes are assumed to be linearly separable. Fig. 1 illustrates the execution output of the LR model.

Fig. 1. Executing the LR model.

```
lr = LogisticRegression()
%time lr.fit(X_train_pca, y_train)

Wall time: 41.1 s

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

B. *Neural Network (NN)*

The NN model utilizes four hidden layers, each has 300 neurons, and one output layer with one neuron. The activation function used for the hidden layers is the Rectified Linear Unit (ReLU). Whereas, the activation function used for the output layer is the Sigmoid function. The used loss function for the model is the binary cross entropy. Fig. 2 illustrates the execution output of the NN model.

Fig. 2. Executing the NN model.

```
model = Sequential()

model.add(Dense(output_dim=300, input_shape=[X_train_pca.shape[1]], activation='relu', \
                W_regularizer=l2(0)))
model.add(Dense(output_dim=300, activation='relu'))
model.add(Dense(output_dim=300, activation='relu'))
model.add(Dense(output_dim=300, activation='relu'))
model.add(Dense(output_dim=1, activation='sigmoid'))
# Compile model
sgd = SGD(lr=0.1)
model.compile(loss='binary_crossentropy', optimizer=sgd)
checkpointer = ModelCheckpoint(filepath='model\\nn\\weights.hdf5', verbose=1, save_best_only=True)
```

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_6 (Dense)              (None, 300)               23100     
_________________________________________________________________
dense_7 (Dense)              (None, 300)               90300     
_________________________________________________________________
dense_8 (Dense)              (None, 300)               90300     
_________________________________________________________________
dense_9 (Dense)              (None, 300)               90300     
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 301       
=================================================================
Total params: 294301
_________________________________________________________________
```

C. *Random Forests (RF)*

In RF, the number of trees in the forest is provided to the cross-validation method as a list of estimators (5, 10, 50, 100, 200, 300, 400, 1000). The bootstrap option is enabled by default. The execution results show that the best number of trees is 1,000. Fig. 3 shows the execution results of the RF algorithm.

Fig. 3. Executing the RF model.

```
n_estimators_list = [5,10,50,100,200,300,400,1000]

rfc = RandomForestClassifier(random_state=47)
rfc_grid = GridSearchCV(estimator=rfc, param_grid=dict(n_estimators=n_estimators_list))
%time rfc_grid.fit(X_train_pca, y_train)

print(rfc_grid)
# summarize the results of the grid search
print(rfc_grid.best_score_)
print(rfc_grid.best_estimator_.n_estimators)
```

```
Wall time: 2h 19min 34s
GridSearchCV(cv=None, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=47, verbose=0, warm_start=False),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'n_estimators': [5, 10, 50, 100, 200, 300, 400, 1000]},
       pre_dispatch='2*n_jobs', refit=True, scoring=None, verbose=0)
0.920046651502
1000
```

D. *Gradient Boosting Classifier (GBC)*

In GBC, the same number of trees, as RF, is provided to the cross-validation method as a list of estimators (5, 10, 50, 100, 200, 300, 400, 1000). The maximum depth option is kept as default (i.e., 3). The execution results show that the best number of trees is 1,000. Fig. 4 shows the execution results of the GBC algorithm.

Fig. 4. Executing the GBC model.

```
gbc = GradientBoostingClassifier(random_state=47)

n_estimators_list = [5,10,50,100,200,300,400,1000]
gbc_grid = GridSearchCV(estimator=gbc, param_grid=dict(n_estimators=n_estimators_list))
%time gbc_grid.fit(X_train_pca, y_train)

print(gbc_grid)
# summarize the results of the grid search
print(gbc_grid.best_score_)
print(gbc_grid.best_estimator_.n_estimators)
```

```
Wall time: 2h 2min 30s
GridSearchCV(cv=None, error_score='raise',
       estimator=GradientBoostingClassifier(init=None, learning_rate=0.1, loss='deviance',
              max_depth=3, max_features=None, max_leaf_nodes=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=100,
              presort='auto', random_state=47, subsample=1.0, verbose=0,
              warm_start=False),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'n_estimators': [5, 10, 50, 100, 200, 300, 400, 1000]},
       pre_dispatch='2*n_jobs', refit=True, scoring=None, verbose=0)
0.903448741181
1000
```

IV. Results

The performance of the four models is discussed in the below sections:

A. *Logistic Regression Performance*

The performance result of the LR model shows TN = 7,460, TP = 15,011, FN = 1,428 and FP = 2,038. This result produces a precision measure for the positive classification (i.e., "Commercials"), which refers to the ratio between the number of TP and the number of total positive predictions (i.e., TP + FP), of 88%.

The result shows a recall measure for the positive classification (i.e., "Commercials"), which refers to the ratio between the number of TP and the number of total actual positives (i.e., TP + FN), of 91%.

Moreover, the result shows that the F1-Score, which combines both precision and recall measures together, reached 80%.

Finally, the result shows that the accuracy of the LR model almost reached 86.64%. Fig. 5 illustrates the performance report of the executed LR model.

Fig. 5. The LR model performance report.

```
print 'accuracy', accuracy_score(y_test, y_predicted_lr)
print 'confusion matrix\n', confusion_matrix(y_test, y_predicted_lr)
print '(row=expected, col=predicted)'
print classification_report(y_test,y_predicted_lr)
```

```
accuracy 0.866368508309
confusion matrix
[[ 7460  2038]
 [ 1428 15011]]
(row=expected, col=predicted)
             precision    recall  f1-score   support

        0.0       0.84      0.79      0.81      9498
        1.0       0.88      0.91      0.90     16439

avg / total       0.87      0.87      0.87     25937
```

B. *Neural Network Performance*

The performance result of the NN model shows TN = 8,622, TP = 15,533, FN = 906 and FP = 876. This result produces a precision measure for the positive classification (i.e., "Commercials") of 95%.

The result shows a recall measure for the positive classification (i.e., "Commercials") of 94%. Moreover, the result shows that the F1-Score reached 95%.

Finally, the result shows that the accuracy of the NN model almost reached 93.13%. Fig. 6 illustrates the performance report of the executed NN model.

Fig. 6. The NN model performance report.

```
print 'accuracy', accuracy_score(y_test, y_predicted_lrmodel)
print 'confusion matrix\n', confusion_matrix(y_test, y_predicted_lrmodel)
print '(row=expected, col=predicted)'
print '\n Classifcation Report'
print classification_report(y_test,y_predicted_lrmodel)
```

```
accuracy 0.93129506111
confusion matrix
[[ 8622   876]
 [  906 15533]]
(row=expected, col=predicted)

 Classifcation Report
             precision    recall  f1-score   support

        0.0       0.90      0.91      0.91      9498
        1.0       0.95      0.94      0.95     16439

avg / total       0.93      0.93      0.93     25937
```

C. RF Performance

The performance result of the RF model shows TN = 8,198, TP = 15,944, FN = 495 and FP = 1,300. This result produces a precision measure for the positive classification (i.e., "Commercials") of 92%.

The result shows a recall measure for the positive classification (i.e., "Commercials") of 97%. Moreover, the result shows that the F1-Score reached 95%.

Finally, the result shows that the accuracy of the RF model almost reached 93.08%. Fig. 7 illustrates the performance report of the executed RF model.

Fig. 7. The RF model performance report.

```
y_predicted_rfc = rfc_grid.predict(X_test_pca)
y_predicted_proba_rfc = rfc_grid.predict_proba(X_test_pca)
y_predicted_proba_rfc = np.asarray(zip(*y_predicted_proba_rfc))[1]

print 'Accuracy: ', accuracy_score(y_test, y_predicted_rfc)
print '\n Decision Tree Results\nConfusion Matrix: '
print confusion_matrix(y_test,y_predicted_rfc)
print '\n Classifcation Report'
print classification_report(y_test,y_predicted_rfc)
```

```
Accuracy:  0.930793846628

 Decision Tree Results
Confusion Matrix:
[[ 8198  1300]
 [  495 15944]]

 Classifcation Report
             precision    recall  f1-score   support

        0.0       0.94      0.86      0.90      9498
        1.0       0.92      0.97      0.95     16439

avg / total       0.93      0.93      0.93     25937
```

D. GBC Performance

The performance result of the GBC model shows TN = 7,966, TP = 15,427, FN = 1,012 and FP = 1,532. This result produces a precision measure for the positive classification (i.e., "Commercials") of 91%.

The result shows a recall measure for the positive classification (i.e., "Commercials") of 94%.

Moreover, the result shows that the F1-Score reached 92%.

Finally, the result shows that the accuracy of the GBC model almost reached 90.19%. Fig. 8 illustrates the performance report of the executed GBC model.

Fig. 8. The GBC model performance report.

```
y_predicted_gbc = gbc_grid.predict(X_test_pca)
y_predicted_proba_gbc = gbc_grid.predict_proba(X_test_pca)
y_predicted_proba_gbc = np.asarray(zip(*y_predicted_proba_gbc))[1]

print 'Accuracy: ', accuracy_score(y_test, y_predicted_gbc)
print '\n Decision Tree Results\nConfusion Matrix: '
print confusion_matrix(y_test,y_predicted_gbc)
print '\n Classifcation Report'
print classification_report(y_test,y_predicted_gbc)
```

```
Accuracy:  0.901916181517

 Decision Tree Results
Confusion Matrix:
[[ 7966  1532]
 [ 1012 15427]]

 Classifcation Report
             precision    recall  f1-score   support

        0.0       0.89      0.84      0.86      9498
        1.0       0.91      0.94      0.92     16439

avg / total       0.90      0.90      0.90     25937
```

V. Conclusion

As shown in Table I, the Neural Networks model achieved the highest performance in terms of precision (i.e., 95%), F1-Score (i.e., 95%) and accuracy (i.e., 93.13%) measures. However, the Random Forests model had the highest recall measure (i.e., 97%). Models Performance Summary

Performance Measure	*LR*	*NN*	*RF*	*GBC*
Precision	88%	95%	92%	91%
Recall	91%	94%	97%	94%
F1-Score	90%	95%	95%	92%
Accuracy	86.64%	93.13%	93.01%	90.19 %

After examining the four models, there is a high confidence that the Neural Networks model, with the previously mentioned structure, can achieve better prediction than the other three models.