VEWS

Snigdha Reddy Obulam (so16g), Swathi Raja (sr16e)

I. INTRODUCTION

Wikipedia is a free internet encyclopedia [2] and is very widely used by everyone in the world as it has almost all the information that we need. But is also compromised by a few vandals, who carry out vandalism acts that involves any addition, removal or change of content in deliberate attempt to compromise the integrity of it [3]. The goal of the project early identification of vandals before any human or known vandalism detection system reports vandalism. Vandalism detection approach goes hand-in-hand with human reporting of vandals. VEWS exploits the differences in the editing behavior of different types of editors to differentiate between them. This can be achieved by statistically analyzing the dataset.

The potential application of this project: Wikipedia's reason is to benefit users by acting as a free encyclopedia, a comprehensive composed digest that contains data on all branches of information inside its five columns. Moreover, while Wikipedia has the potential to be a key asset in schools at all levels due to its breadth, by and large quality, and free accessibility - the chance of uncovering children to unseemly fabric has been a deterrent to selection. Moreover, the nearness of vandalism has made it troublesome to create inactive, high-quality previews of Wikipedia substance, such as those that the Wikipedia 1.0 project or WikiReader plans to convey in creating nations with destitute Web network. This reason can be accomplished as it were when it is free from vandalized articles or any enlisted vandal client.

II. EXISTING APPROACHES

The goal of this project is to identify vandals before any human or any vandalism detection system reports vandalism to Wikipedia administrators. Current Wikipedia vandal detection system has focused on problem of identifying pages whose text has been damaged. Current Wikipedia vandalism detection tools are ClueBot NG [4] and STiki [5].

Cluebot NG is Wikipedia's most dynamic anti-vandal bot, and as of now returns over 9000 vandal alerts a day on Wikipedia. Existing anti-vandal bots use straightforward inactive heuristics that capture a moderately little parcel of vandalism and with an unsatisfactory wrong positive rate. Cluebot-NG offers no code with the unique cluebot and employments totally different algorithms to detect vandalism. Cluebot-NG requires a dataset to operate, the dataset can be utilized to provide reasonably accurate measurements on its accuracy and operation. Diverse parts are utilized for training and testing, so these calculations are not biased. Cluebot-NG uses a mix of vandalism detection methods which uses machine learning as core. It uses machine learning basics, bayseian classifiers, artificial neural network, threshold calculation, post-processing filters [4].

STiki is a device utilized to distinguish and return vandalism or other unconstructive edits on Wikipedia, accessible to trusted clients. STiki chooses edits to appear to end clients; in case a shown edit is judged to be vandalism, STiki at that point streamlines the inversion and caution handle. STiki encourages collaboration in returning vandalism; centrally put away records of alters to be reviewed are served to STiki clients to decrease repetitive exertion. STiki is not a Wikipedia bot: it is a brilliantly steering device that coordinates human clients to potential vandalism for authoritative classification. The "metadata framework" analyzes as it were four fields of an edit when scoring: (1) timestamp, (2) editor, (3) article, and (4) revision comment. These fields are utilized to calculate highlights relating to the editor's enlistment status, alter time-of-day, alter day-of-week, topographical root, page history, category participations, amendment comment length, etc. These signals are given to an AD-Tree classifier to reach at vandalism probabilities. The ML models are trained over classifications given on the STiki frontend [5].

III. METHODOLOGY

In this section, we factually analyze editing behaviors of vandals and benign users in order to distinguish behavioral similarities and differences.

1. Benign and Vandal Users Similarities

a. Both vandals and benign users are much more likely to re-edit a page compared to editing a new page [1].
b. Both vandals and benign users consecutively edit the same page quickly [1].
c. Both vandals and benign users exhibit similar navigation patterns [1].
d. In their first few edits, both vandals and benign users have similar editing behavior [1].

2. Benign and Vandal Users Differences

a. Vandals make faster edits than benign users [1].
b. Benign users spend more time editing a new (to them) page than vandals [1].
c. The probability that benign users edit a meta-page is much higher than the same probability in the case of vandals [1].

In conclusion, vandals make faster edits than benign users and vandals are much less engaged in edits of meta-page (discussions with community).

3. Wikipedia Vandal Behavior Approach (WVB)

All features utilized for vandal prediction are behavior based and incorporate no human created return data at all. In this way, this approach frames an early warning framework for Wikipedia administrators. WVB uses features determined from successive edits. These are found by frequent mining of the dataset. Particularly, we extricate the visit designs on both benign and vandal client logs – at that point, for each frequent pattern of benign clients, we compute the recurrence of the same design for vandals and vice versa. At long last, we select the designs having critical recurrence distinction between the two classes. The features are described below [1]:

a. Consecutive re-edit slowly (crs): Whether the user edited or not the same page consecutively with time-gap more than 15 minutes.
b. Consecutive re-edit very fast (crv): Whether the user edited or not the same page consecutively with time-gap less than 3 minutes.
c. Consecutive re-edit of a meta-page (crm): Number of times the user re-edited meta-page consecutively.
d. Consecutive re-edit of a non-meta-page (crn): Whether the user edited or not the same meta-page consecutively.
e. Consecutive re-edit of a meta-page very fast (crmv): Whether the user edited or not the same meta-page consecutively with time-gap less than 3 minutes.
f. Consecutive re-edit of a meta-page fast (crmf): Whether the user edited or not the same meta-page consecutively with time-gap between 3 minutes to 15 minutes.

g. Consecutive re-edit of a meta-page slowly (crms): Whether the user edited or not the same meta-page consecutively with time-gap more than 15 minutes.

h. Consecutively re-edit fast and consecutively re-edit very fast (crf_crv): Whether or not user re-edited the same article within 15 minutes, and later re-edited an article and less than 3 minutes passed between the second pair of edits.

i. First edit meta-page (fm): whether the first edit of the user was on a meta-page or not.

j. Edit of a new page at distance at most 3 hops slowly (ntus): Whether or not the user edited a new page which is within 3 hops or less of the previous page and the time gap between the two edits exceeds 15 minutes.

k. Edit of a new page at distance at most 3 hops slowly and twice (nts_nts): Whether or not the user edited a new page such that new page can be reached from previous page link-wise with at most 3 hops, and more than 15 minutes passed between the edit of two pages.

The importance of features described are calculated. Feature *fm* tells if the first page edited by the user is a non-meta page then the user is more likely to be a vandal than benign user. Benign users are likely to take longer time than vandals to edit a new page. Vandal users are less likely to re-edit the meta page. These features indicate editing meta-pages versus normal pages which is a strong indicator whether the user is benign or vandal. The most common target of vandal users are normal pages whereas benign users discuss about the content with other users, which is done on meta-page.

IV. IMPLEMENTATION

The datasets used here was collected from January 01, 2013 and July 31, 2014 containing all registered benign and vandal users. Datasets are available in [6]. Users.csv dataset contains list of all registered users in the format (user name, user id, blocked time, blocked reason, type). Benign_year_month and vandal_year_month dataset contains edits made by benign users and vandal users respectively. The format of benign and vandal user dataset is (username, revid, revtime, page_title, isReverted, reverttime, cluebotRevert, stiki_score, stiki_REP_USER). The importance of features described in the previous section are calculated.

We have used Python 2.7 for the implementation part. The first step in implementation is preprocessing of benign and vandal user datasets. In the preprocessing stage, benign and vandal user datasets are combined. By using the most descriptive features which include fm, ntus, crmv, crmf, crms and crm that mainly deals with meta pages, it detects whether the user is benign or a vandal for each feature conditions. These descriptive features are concatenated to the combined dataset. Now, the preprocessed dataset is in the format (vandal, username, revid, revtime, page_title, isReverted, reverttime, ntus, fm, crmv, crmf, crms, crm). The preprocessed dataset is now shuffled for randomized data.

In the training phase, the preprocessed dataset is split into training and test dataset. As the range of values in dataset vary, scaling is performed on dataset using Min-Max scaler. The classifiers used for training are decision tree classifier, random forest classifier, MLP classifier, logistic regression, SVM and gaussian naïve bayes. Decision Trees are a non-parametric supervised learning strategy used for classification. The objective is to make a model that predicts the value of a target variable by learning basic decision rules induced from the information features [7]. A random forest is a meta estimator that fits several decision tree classifiers on different sub-samples of the dataset and utilize averaging to progress the prescient accuracy and control over-fitting [8]. A multilayer perceptron (MLP) is a course of feedforward artificial neural network. An MLP comprises of at least three layers of nodes. MLP utilizes a supervised learning procedure called backpropagation for training. It's different layers and non-linear activation recognizes MLP from a linear perceptron [9]. Logistic regression is a regression model where the dependent variable is categorical. This model is used to estimate the probability of binary response based on predictor features [10]. Support Vector Machines (SVM) are supervised learning models used for classification and regression analysis [11]. Gaussian Naive Bayes classifiers are a family of straightforward probabilistic classifiers based on applying Bayes' hypothesis with strong independence assumptions between the features [12].

The entire dataset is split into training and test dataset for performing the cross validation and to train the classifier models. The test dataset is used for prediction using classifiers mentioned above. The trained model for each classifier is encoded and stored for future to reduce learning every time for prediction. The most challenging part in the implementation was to come up with idea on how to compute the features and use them for predicting the users. As beginners in Python, there was a lot to learn from working with data-frames, arrays to sklearn methodologies. It was also challenging to understand the process of encoding the training model and using it to predict the test dataset in the future.

V. EXPERIMENTAL RESULTS

After preprocessing, we obtain the processed dataset. This processed dataset is then trained by cross validating and the model is stored for future predictions.

Prediction accuracies:

```
**************Running trained model DecisionTreeClassifier***************
loading learned model...
loaded model..
Time to predict test data:  1.0  minutes
Predicting accuracy:  0.636790098812
confusion matrix:
[[10506  5647]
 [ 1631  2254]]
classification report:
             precision    recall  f1-score   support

          0       0.87      0.65      0.74     16153
          1       0.29      0.58      0.38      3885

avg / total       0.75      0.64      0.67     20038

**************Running trained model RandomForestClassifier***************
loading learned model...
loaded model..
Time to predict test data:  1.0  minutes
```

```
Predicting accuracy: 0.739694580297
confusion matrix:
[[13172  2981]
 [ 2235  1650]]
classification report:
             precision    recall  f1-score   support

          0       0.85      0.82      0.83     16153
          1       0.36      0.42      0.39      3885

avg / total       0.76      0.74      0.75     20038

**************Running trained model MLPClassifier***************
loading learned model...
loaded model..
Time to predict test data:  1.0  minutes
Predicting accuracy:  0.830721628905
confusion matrix:
[[15976   177]
 [ 3215   670]]
classification report:
             precision    recall  f1-score   support

          0       0.83      0.99      0.90     16153
          1       0.79      0.17      0.28      3885

avg / total       0.82      0.83      0.78     20038

**************Running trained model LogisticRegression***************
loading learned model...
loaded model..
Time to predict test data:  0.0  minutes
Predicting accuracy:  0.830172671923
confusion matrix:
[[15981   172]
 [ 3231   654]]
classification report:
             precision    recall  f1-score   support

          0       0.83      0.99      0.90     16153
          1       0.79      0.17      0.28      3885

avg / total       0.82      0.83      0.78     20038

**************Running trained model GaussianNB***************
loading learned model...
```

```
loaded model..
Time to predict test data:  1.0  minutes
Predicting accuracy:  0.81959277373
confusion matrix:
[[16153     0]
 [ 3615   270]]
classification report:
             precision    recall  f1-score   support

          0       0.82      1.00      0.90     16153
          1       1.00      0.07      0.13      3885

avg / total       0.85      0.82      0.75     20038

**************Running trained model SVM***************
loading learned model...
loaded model..
Time to predict test data:  1.0  minutes
Predicting accuracy:  0.813853678012
confusion matrix:
[[16153     0]
 [ 3730   155]]
classification report:
             precision    recall  f1-score   support

          0       0.81      1.00      0.90     16153
          1       1.00      0.04      0.08      3885

avg / total       0.85      0.81      0.74     20038
```

Result comparison:

From comparison, we can assess that Decision Tree classifier and Random Forest classifier have high performance while training and while, predicting the accuracies have reduced, this may be due to overfitting the training dataset and not performing well on unseen data.

Classifier	Training set validation accuracy	Predicting test set
Decision tree	96.19%	63.67%
Random Forest	95.77%	73.96%
Multi-layer perceptron	83.38%	83.07%
Gaussian NB	82.10%	81.95%
Logistic Regression	83.37%	83.01%
SVM	81.31%	81.3%

Table 1:Results comparison

VI. CONCLUSION

The WVB approach used in this project uses edit-pattern of users to study the behavior of vandals on Wikipedia and distinguish these features from those of benign users. Detailed analysis of behavior to distinguish vandals from benign users include: Benign users first edit are most likely to occur on meta-page whereas first edits of vandal users more likely to occur on normal pages. Benign user takes longer time to edit a page than any vandal user. Benign users are more likely to re-edit the same page within 3 minutes compared to vandal users because benign users want to improve something they previously edited. These are the features that distinguishes vandal users from benign users. We used the decision tree classifier, random forest classifier, MLP classifier, logistic regression and SVM classifiers while training a model. By comparing the WVB approach with the state of the art tools, 'STiki' and 'cluebot-NG' the resulting prediction accuracies from **WVB (83.07%) approach is better than STiki (75%) and cluebot-NG (71.4%)** [1] and WVB approach does not require any human generated revert information. So, this approach can be used by Wikipedia administrators as an early warning system.

REFERENCES

[1] S. Kumar, F. Spezzano, V. S. Subrahmanian, *"VEWS: A wikipedia vandal early warning system"*. Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2015.

[2] http://en.wikipedia.org/wiki/Encyclopedia

[3] http://en.wikipedia.org/wiki/Wikipedia:Vandalism

[4] https://en.wikipedia.org/wiki/User:ClueBot_NG

[5] https://en.wikipedia.org/wiki/Wikipedia:STiki

[6] http://www.cs.umd.edu/~vs/vews

[7] http://scikit-learn.org/stable/modules/tree.html

[8] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[9] http://scikit-learn.org/stable/modules/neural_networks_supervised.html

[10] http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[11] http://scikit-learn.org/stable/modules/svm.html

[12] http://scikit-learn.org/stable/modules/naive_bayes.html

Source Code

Preprocessing

```
import math
import os
import time
import pandas as pd
from sklearn.utils import shuffle


def calculate_crm_feature(row, data):
    vandal = 0
    d = data.loc[(data['username'] == row['username']) & (data['revtime'] <= row['revtime'])]
    d.sort_values(['username', 'revtime'], ascending=True)
    if len(d) > 1:
        last_edited_page = d.iloc[[d.shape[0]-1]]
        if is_meta(last_edited_page.iloc[0]['pagetitle'].lower()) and is_meta(row['pagetitle'].lower())
and last_edited_page.iloc[0]['pagetitle'] == row['pagetitle']:
            vandal = 0
        else:
            vandal = 1
        return vandal
    return vandal



def calculate_fm_feature(row, data):
    vandal = 0
    # checks feature fm
    d = data.loc[(data['username'] == row['username']) and (data['revtime'] <= row['revtime'])]
    d.sort_values(['username', 'revtime'], ascending=True)

    for r in d.iterrows():
        if is_meta(r['pagetitle'].lower()):
            vandal = 0
        else:
            vandal = 1
        return vandal
    return vandal


def calculate_crm_vfs_feature(row, data):
    d = data.loc[(data['username'] == row['username']) & (data['revtime'] <= row['revtime'])]
    d.sort_values(['username', 'revtime'], ascending=True)


    three_min = 180000
    fifteen_min = 900000
    if len(d) > 1:
        last_edited_page = d.iloc[[d.shape[0]-1]]
        if  is_meta(last_edited_page.iloc[0]['pagetitle'].lower())  &  is_meta(row['pagetitle'].lower())
and last_edited_page.iloc[0]['pagetitle'] == row['pagetitle']:
                interval = row['revtime'] - last_edited_page.iloc[0]['revtime']


                if interval < three_min:
                    return 0, 1, 1
                elif interval >= three_min & interval <= fifteen_min:
                    return 1, 0 , 1
                else:
                    return 1, 1, 0
    return 0, 0, 0
def calculate_ntus_feature(row, data):
    #ntus :Whether or not the user edited a new page which is within 3 hops or less of the previous
page and the time gap between the two edits exceeds 15 minutes.
    vandal = 0
```

```
d = data.loc[(data['username'] == row['username']) & (data['revtime'] <= row['revtime'])]
d.sort_values(['username', 'revtime'], ascending=True)
fifteen_min = 900000
pagetitle = row['pagetitle']
hop = len(d)
if hop > 1:
    last_edited_page = d.iloc[[d.shape[0]-1]]
    hop_time_diff = row['revtime'] - last_edited_page.iloc[0]['revtime']

    if last_edited_page.iloc[0]['pagetitle'] != pagetitle and hop_time_diff >= fifteen_min:
        vandal = 0
    else:
        vandal = 1
    return vandal

if hop > 2:
    second_last_page = d.iloc[[d.shape[0]-2]]
    hop_time_diff = row['revtime'] - second_last_page.iloc[0]['revtime']

    if second_last_page.iloc[0]['pagetitle'] != pagetitle and hop_time_diff >= fifteen_min:
        vandal = 0
    else:
        vandal = 1
    return vandal

if hop > 3:
    third_last_page = d.iloc[[d.shape[0]-3]]
    hop_time_diff = row['revtime'] - third_last_page.iloc[0]['revtime']

    if third_last_page.iloc[0]['pagetitle'] != pagetitle and hop_time_diff >= fifteen_min:
            vandal = 0
        else:
            vandal =1
        return vandal
    return vandal

def is_meta(page):
    if page.startswith('user:') or page.startswith('user talk:') or page.startswith('talk:'):
        return 1
    else:
        return 0

def convert_to_millis(row, column):
    if row[column] == '-':
        return -1
    timestamp = pd.Timestamp(row[column])
    if type(timestamp) == pd.Timestamp:
        timestamp = pd.Timestamp(row[column])
        return timestamp.value
    else:
        return -1

def perform_preprocessing(merged_data, data):
    # sorting data on page title ,uname, rev time
    print "Sorting data"
    merged_data.sort_values(['pagetitle', 'username', 'revtime'], ascending=True)
    # computing features
    print "fm"
    merged_data['fm'] = merged_data.apply(lambda row: calculate_fm_feature(row, data), axis=1)
    print "crm vfs"
    merged_data[['crmv', 'crmf', 'crms']] = merged_data.apply(lambda row:
pd.Series(calculate_crm_vfs_feature(row, data)), axis=1)
    print "ntus"
    merged_data['ntus'] = merged_data.apply(lambda row: calculate_ntus_feature(row, data),
axis=1)
    print "crm"
    merged_data['crm'] = merged_data.apply(lambda row: calculate_crm_feature(row, data),
axis=1)

    # randomizing
    merged_data = shuffle(merged_data)
    write_to_file(merged_data)

def write_to_file(merged_data):
    # storing the preprocessed dataset
    print "storing preprocessed data."
    merged_data.to_csv('processed_data.csv', sep=',', encoding='utf-8', index=False)


def main():
    # access benign and vandal datasets
    #start = time.time()
    directory = os.path.join("vews_dataset_v1.1/")
    benign_data = pd.DataFrame()
    vandal_data = pd.DataFrame()
    for root, dirs, files in os.walk(directory):
        for file_name in files:
            if file_name.endswith(".csv"):
                data = pd.read_csv(root + file_name, sep=',', usecols=['username', 'revid', 'revtime',
'pagetitle',
                                                    'isReverted', 'revertTime'])
            data['revtime'] = data.apply(lambda row: convert_to_millis(row, 'revtime'), axis=1)
            data['revertTime'] = data.apply(lambda row: convert_to_millis(row, 'revertTime'), axis=1)
            if file_name.startswith("benign"):
                data['vandal'] = 0
                benign_data = benign_data.append(data, ignore_index=True)
            elif file_name.startswith('vandal'):
                data['vandal'] = 1
                vandal_data = vandal_data.append(data, ignore_index=True)

    # Merge benign and vandal data
    print "Merging benign and vandal datasets"
    merged_data = benign_data.append(vandal_data, ignore_index=True)
    perform_preprocessing(merged_data, data)

main()
```

Training

```
import cPickle
from collections import defaultdict
import math
import time
from sklearn import svm, preprocessing
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import LabelEncoder



training_data = pd.read_csv('processed_data.csv', delimiter=',')
#print " dataset size: ", training_data.shape

# encoding user,page features to numeric values
encoder = LabelEncoder()
training_data['username'] = encoder.fit_transform(training_data['username'])
training_data['pagetitle'] = encoder.fit_transform(training_data['pagetitle'])

# storing data in numpy array
xt = training_data.as_matrix(columns=['username', 'revtime', 'pagetitle', 'ntus', 'fm', 'crmv', 'crmf' ,
'crms', 'crm'])
yt = training_data.as_matrix(columns=['vandal'])
# cross validation
x_train, x_test, y_train, y_test = train_test_split(xt, yt)

#scaling datasets
scaler = preprocessing.MinMaxScaler(feature_range=(0,1)).fit(x_train)
xt_train = scaler.transform(x_train)
xt_test = scaler.transform(x_test)

classifiers = []
classifiers.append(("DecisionTreeClassifier", DecisionTreeClassifier()))
classifiers.append(("RandomForestClassifier", RandomForestClassifier()))
classifiers.append(("MLPClassifier", MLPClassifier(hidden_layer_sizes=(8,8,8))))
classifiers.append(("LogisticRegression", LogisticRegression()))
classifiers.append(("GaussianNB", GaussianNB()))
#svm is running only with a smaller dataset so trained with a smaller data and stored
#classifiers.append(("SVM", svm.SVC()))
for cls in classifiers:
    print "****************Running classifier : %s****************************" % cls[0]
    start = time.time()
    classifier = cls[1].fit(xt_train, y_train.ravel())
    end = time.time()
    print "Time for training: ", math.ceil((end - start)/60), " min"
    start = time.time()
    y_p = classifier.predict(xt_test)
    end = time.time()
    print "Time for predicting:", math.ceil((end - start)/60), " min"

    # prediction metrics
    print 'Prediction Accuracy :', accuracy_score(y_test, y_p)
    print "confusion matrix:"
    print(confusion_matrix(y_test, y_p))
    print "classification report:"
    print(classification_report(y_test, y_p))

#    #roc_curve
#    print ('ROC',roc_auc_score(y_test,y_p))
#    fpr, tpr, thresholds = roc_curve(y_test,y_p)
#    print "fpr", fpr
#    print "tpr", tpr
#    plt.title('Receiver Operating Characteristic')
#    plt.plot(fpr,tpr)
#    plt.show()

    # storing the model for future predictions
    loc = "trained_model_%s.obj"%cls[0]
    f = file(loc,"wb")
    cPickle.dump(classifier, f, protocol=cPickle.HIGHEST_PROTOCOL)
    f.close()
```

Prediction

```
import cPickle
import math
import pandas as pd
import time
from sklearn import preprocessing
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import LabelEncoder

models = []
models.append("DecisionTreeClassifier")
models.append("RandomForestClassifier")
models.append("MLPClassifier")
models.append("LogisticRegression")
models.append("GaussianNB")
models.append("SVM")

for model in models:
    print "**************Running trained model %s**************"% model
    loc = "trained_model_%s.obj"%model
    print 'loading learned model...'
    f = file(loc,"rb") # Specify file location
    clf = cPickle.load(f)
    f.close()
    print 'loaded model..'

    test_data = pd.read_csv('test.csv', delimiter=',')

    # encoding user,page features to numeric values
    encoder = LabelEncoder()
    test_data['username'] = encoder.fit_transform(test_data['username'])
    test_data['pagetitle'] = encoder.fit_transform(test_data['pagetitle'])

    xt = test_data.as_matrix(columns=['username', 'revtime', 'pagetitle', 'ntus', 'fm', 'crmv', 'crmf' ,
'crms', 'crm'])
    yt = test_data.as_matrix(columns=['vandal'])

    # scaling testing data
    scaler = preprocessing.MinMaxScaler(feature_range=(0,1)).fit(xt)
    xtt = scaler.transform(xt)

    # Predict on test data
    start = time.time()
    y_p = clf.predict(xtt)
    end = time.time()
    #print y_p
    print "Time to predict test data: ", math.ceil((end - start)/60), " minutes"

    # prediction metrics
    print 'Prediction accuracy : ', accuracy_score(yt, y_p)
    print "confusion matrix:"
    print(confusion_matrix(yt, y_p))
    print "classification report:"
```

```
print(classification_report(yt, y_p))
```