```
import numpy as np
import pandas as pd
import sklearn as sklearn
import sklearn.decomposition as pca
from time import time
from IPython.display import display # Allows the use of display() for DataFrames
%matplotlib inline
# Load the online news dataset
data = pd.read_csv("/content/OnlineNewsPopularity.csv")
display(data.head())
```

	url	timedelta	n_tokens_title	n_tokens_content	n_
0	http://mashable.com/2013/01/07/amazon-instant-...	731.0	12.0	219.0	
1	http://mashable.com/2013/01/07/ap-samsung-spon...	731.0	9.0	255.0	
2	http://mashable.com/2013/01/07/apple-40-billio...	731.0	9.0	211.0	
3	http://mashable.com/2013/01/07/astronaut-notre...	731.0	9.0	531.0	
4	http://mashable.com/2013/01/07/att-u-verse-apps/	731.0	13.0	1072.0	

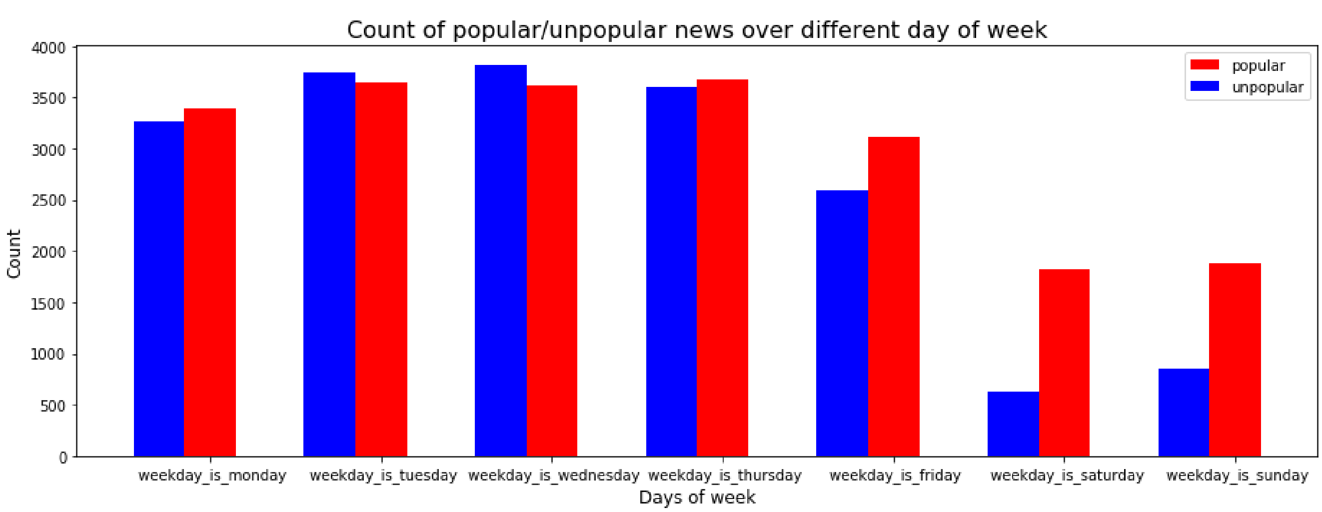
5 rows × 61 columns

```
# Get the statistics of original target attribute
popularity_raw = data[data.keys()[-1]]
popularity_raw.describe()
# Encode the label by threshold 1400
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
popular_label = pd.Series(label_encoder.fit_transform(popularity_raw>=1400))
```

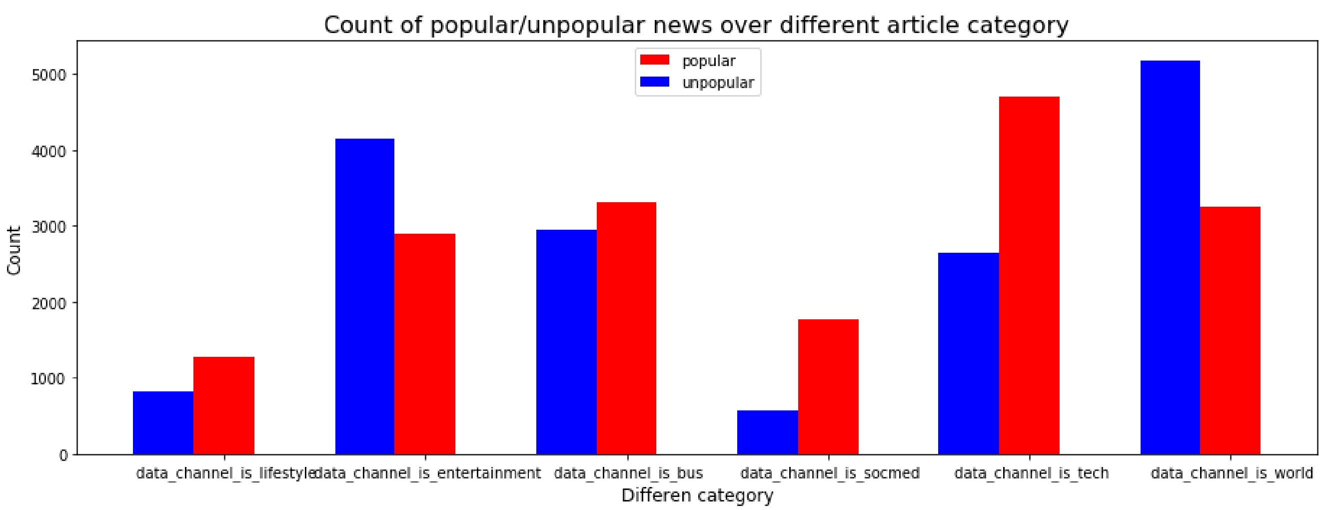
```
# Get the features from dataset
features_raw = data.drop(['url',data.keys()[1],data.keys()[-1]], axis=1)
display(features_raw.head())
```

	n_tokens_title	n_tokens_content	n_unique_tokens	n_non_stop_words	n_non_stop_unic
0	12.0	219.0	0.663594	1.0	
1	9.0	255.0	0.604743	1.0	

```
# Visualize the feature of different day of week
columns_day = features_raw.columns.values[29:36]
unpop=data[data[' shares']<1400]
pop=data[data[' shares']>=1400]
unpop_day = unpop[columns_day].sum().values
pop_day = pop[columns_day].sum().values

import matplotlib.pyplot as pl
from IPython import get_ipython
get_ipython().run_line_magic('matplotlib', 'inline')

fig = pl.figure(figsize = (13,5))
pl.title("Count of popular/unpopular news over different day of week", fontsize = 16)
pl.bar(np.arange(len(columns_day)), pop_day, width = 0.3, align="center", color = 'r', \
          label = "popular")
pl.bar(np.arange(len(columns_day)) - 0.3, unpop_day, width = 0.3, align = "center", color = '
          label = "unpopular")
pl.xticks(np.arange(len(columns_day)), columns_day)
pl.ylabel("Count", fontsize = 12)
pl.xlabel("Days of week", fontsize = 12)

pl.legend(loc = 'upper right')
pl.tight_layout()
pl.savefig("days.pdf")
pl.show()
```



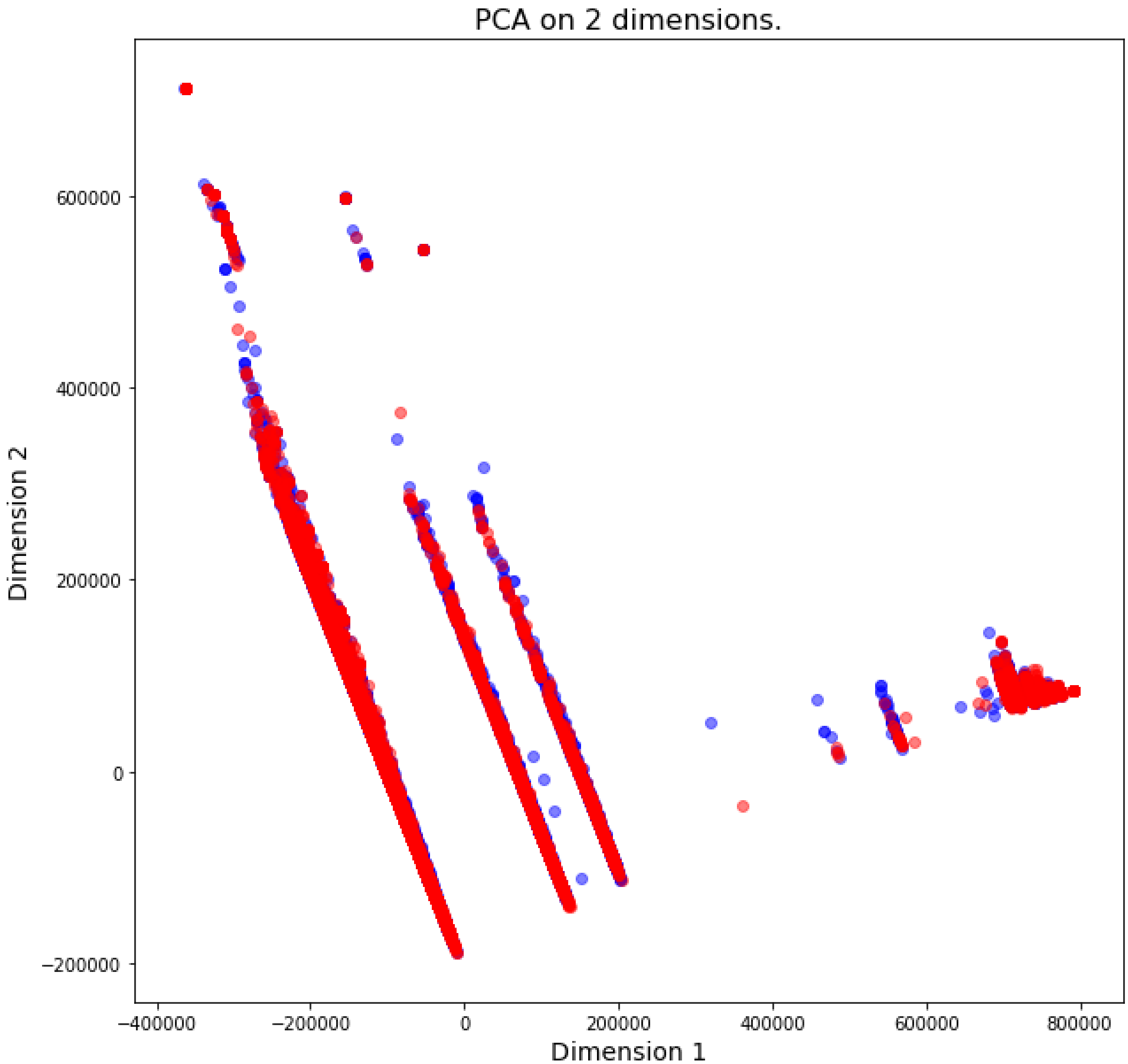
```
# Visualize the feature of different article category
columns_chan=features_raw.columns.values[11:17]
unpop_chan = unpop[columns_chan].sum().values
pop_chan = pop[columns_chan].sum().values
fig = pl.figure(figsize = (13,5))
pl.title("Count of popular/unpopular news over different article category", fontsize = 16)
pl.bar(np.arange(len(columns_chan)), pop_chan, width = 0.3, align="center", color = 'r', \
          label = "popular")
pl.bar(np.arange(len(columns_chan)) - 0.3, unpop_chan, width = 0.3, align = "center", color =
          label = "unpopular")
pl.xticks(np.arange(len(columns_chan)), columns_chan)

pl.ylabel("Count", fontsize = 12)
pl.xlabel("Differen category", fontsize = 12)

pl.legend(loc = 'upper center')
pl.tight_layout()
pl.savefig("chan.pdf")
pl.show()
```

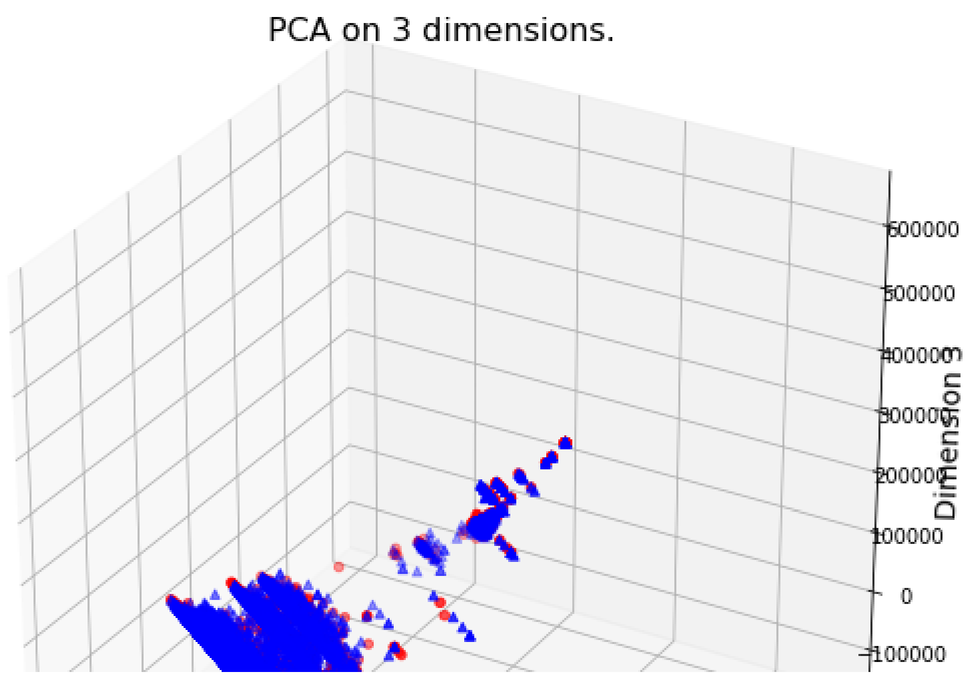


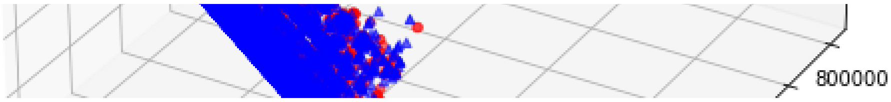

```
# Normalize the numerical features
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
numerical = [' n_tokens_title', ' n_tokens_content', ' num_hrefs', ' num_self_hrefs', ' num_i
             ' average_token_length',' num_keywords',' self_reference_min_shares',' self_refer
              ' self_reference_avg_sharess']
features_raw[numerical] = scaler.fit_transform(data[numerical])
display(features_raw.head(n = 1))
```

	n_tokens_title	n_tokens_content	n_unique_tokens	n_non_stop_words	n_non_stop_unic
0	0.47619	0.025844	0.663594	1.0	

```
# PCA
from sklearn.decomposition import PCA
pca = PCA(n_components=2).fit(features_raw)
reduced_features = pca.transform(features_raw)
reduced_features = pd.DataFrame(reduced_features, columns = ['Dimension 1', 'Dimension 2'])
reduced_features_pop = reduced_features[data[' shares']>=1400]
reduced_features_unpop = reduced_features[data[' shares']<1400]


fig, ax = pl.subplots(figsize = (10,10))
# Scatterplot of the reduced data
ax.scatter(x=reduced_features_pop.loc[:, 'Dimension 1'], y=reduced_features_pop.loc[:, 'Dimer
           c='b',alpha=0.5)
ax.scatter(x=reduced_features_unpop.loc[:, 'Dimension 1'], y=reduced_features_unpop.loc[:, 'D
           c='r', alpha=0.5)
ax.set_xlabel("Dimension 1", fontsize=14)
ax.set_ylabel("Dimension 2", fontsize=14)
ax.set_title("PCA on 2 dimensions.", fontsize=16);
pl.savefig("pca2.jpg")

from mpl_toolkits.mplot3d import Axes3D
pca = PCA(n_components=3).fit(features_raw)
reduced_features = pca.transform(features_raw)
reduced_features = pd.DataFrame(reduced_features, columns = ['Dimension 1', 'Dimension 2','Di
reduced_features_pop = reduced_features[data[' shares']>=1400]
reduced_features_unpop = reduced_features[data[' shares']<1400]
# 3D scatterplot of the reduced data
fig = pl.figure(figsize = (10,10))
ax = fig.add_subplot(111, projection='3d')
ax.scatter( reduced_features_pop.loc[:, 'Dimension 2'],reduced_features_pop.loc[:, 'Dimension
           reduced_features_pop.loc[:, 'Dimension 3'], c='b',marker='^')
ax.scatter(reduced_features_unpop.loc[:, 'Dimension 2'],reduced_features_unpop.loc[:, 'Dimens
           reduced_features_unpop.loc[:, 'Dimension 3'], c='r')
ax.set_xlabel("Dimension 2", fontsize=14)
ax.set_ylabel("Dimension 1", fontsize=14)
ax.set_zlabel("Dimension 3", fontsize=14)
ax.set_title("PCA on 3 dimensions.", fontsize=16);
pl.savefig("pca3.jpg")
```


PCA on 2 dimensions.
Dimension 2
Dimension 1
600000
400000
200000
0
−200000
−400000
−200000
0
200000
400000
600000
800000


PCA on 3 dimensions.
Dimension 3
600000
500000
400000
300000
200000
100000
0
−100000

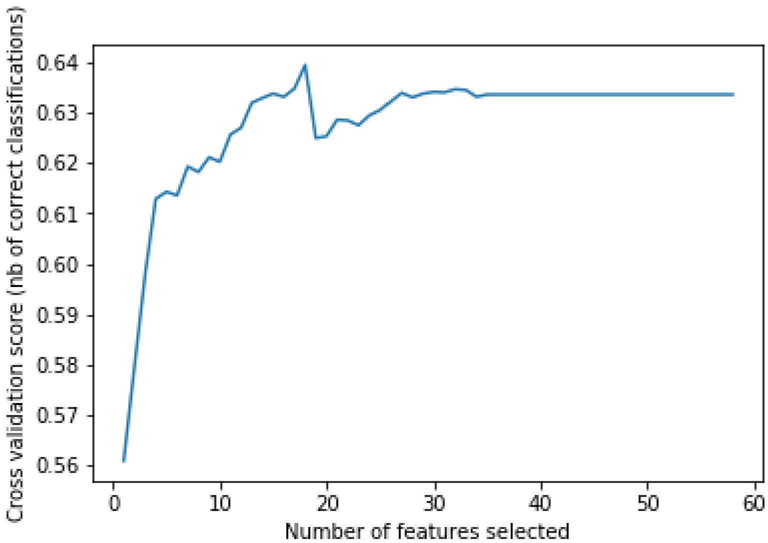



```
from sklearn.feature_selection import RFECV
from sklearn.svm import SVR
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier

estimator = AdaBoostClassifier(random_state=0)
selector = RFECV(estimator, step=1, cv=5)
selector = selector.fit(features_raw, popular_label)
selector.ranking_

estimator_LR = LogisticRegression(random_state=0)
selector_LR = RFECV(estimator_LR, step=1, cv=5)
selector_LR = selector_LR.fit(features_raw, popular_label)
selector_LR.ranking_

estimator_RF = RandomForestClassifier(random_state=0)
selector_RF = RFECV(estimator_RF, step=1, cv=5)
selector_RF = selector_RF.fit(features_raw, popular_label)
selector_RF.ranking_
```

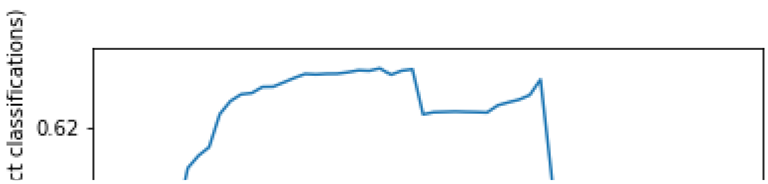
```
/usr/local/lib/python2.7/dist-packages/sklearn/linear_model/logistic.py:433: FutureWarn
  FutureWarning)
/usr/local/lib/python2.7/dist-packages/sklearn/ensemble/forest.py:246: FutureWarning: T
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
array([1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

```
pl.figure()
pl.xlabel("Number of features selected")
pl.ylabel("Cross validation score (nb of correct classifications)")
pl.plot(range(1, len(selector.grid_scores_) + 1), selector.grid_scores_)
pl.savefig('RFE_ADA.pdf')
pl.show()

print features_raw.columns.values[selector.ranking_==1].shape[0]
print features_raw.columns.values[selector.ranking_==1]
features_ADA = features_raw[features_raw.columns.values[selector.ranking_==1]]

pl.figure()
pl.xlabel("Number of features selected")
pl.ylabel("Cross validation score (nb of correct classifications)")
pl.plot(range(1, len(selector_LR.grid_scores_) + 1), selector_LR.grid_scores_)
pl.savefig('RFE_LR.pdf')
pl.show()
```

```
print features_raw.columns.values[selector_LR.ranking_==1].shape[0]
print features_raw.columns.values[selector_LR.ranking_==1]
features_LR = features_raw[features_raw.columns.values[selector_LR.ranking_==1]]

pl.figure()
pl.xlabel("Number of features selected")
pl.ylabel("Cross validation score (nb of correct classifications)")
pl.plot(range(1, len(selector_RF.grid_scores_) + 1), selector_RF.grid_scores_)
pl.savefig('RFE_RF.pdf')
pl.show()

print features_raw.columns.values[selector_RF.ranking_!=1].shape[0]
print features_raw.columns.values[selector_RF.ranking_!=1]
features_RF = features_raw[features_raw.columns.values[selector_RF.ranking_==1]]
```




```
18
[' n_non_stop_unique_tokens' ' num_self_hrefs' ' num_imgs' ' num_videos'
 ' num_keywords' ' data_channel_is_entertainment'
 ' data_channel_is_socmed' ' kw_min_min' ' kw_avg_min' ' kw_min_max'
 ' kw_max_max' ' kw_min_avg' ' kw_max_avg' ' kw_avg_avg'
 ' self_reference_min_shares' ' self_reference_max_shares'
 ' global_subjectivity' ' global_sentiment_polarity']
```



```
# Split data into training and testing sets
from sklearn.metrics import accuracy_score, fbeta_score, roc_curve, auc, roc_auc_score
from sklearn.model_selection import train_test_split


X_train_ADA, X_test_ADA, y_train_ADA, y_test_ADA = train_test_split(features_ADA, popular_lab

X_train_LR, X_test_LR, y_train_LR, y_test_LR = train_test_split(features_LR, popular_label, t

X_train_RF, X_test_RF, y_train_RF, y_test_RF = train_test_split(features_RF, popular_label, t

print "Training set has {} samples.".format(X_train_ADA.shape[0])
print "Testing set has {} samples.".format(X_test_ADA.shape[0])
```

```
Training set has 35679 samples.
Testing set has 3965 samples.
```

```
def train_predict(learner, sample_size, X_train, y_train, X_test, y_test):
    '''
    inputs:
       - learner: the learning algorithm to be trained and predicted on
       - sample_size: the size of samples (number) to be drawn from training set
       - X_train: features training set
       - y_train: income training set
```

```
       - X_test: features testing set
       - y_test: income testing set
    '''

    results = {}

    start = time() # Get start time
    learner.fit(X_train[:sample_size], y_train[:sample_size])
    end = time() # Get end time

    results['train_time'] = end-start

    # Get predictions on the first 4000 training samples
    start = time() # Get start time
    predictions_test = learner.predict(X_test)
    predictions_train = learner.predict(X_train[:4000])
    end = time() # Get end time

    # Calculate the total prediction time
    results['pred_time'] = end-start

    # Compute accuracy on the first 4000 training samples
    results['acc_train'] = accuracy_score(y_train[:4000],predictions_train)

    # Compute accuracy on test set
    results['acc_test'] = accuracy_score(y_test,predictions_test)

    # Compute F-score on the the first 4000 training samples
    results['f_train'] = fbeta_score(y_train[:4000],predictions_train,beta=1)

    # Compute F-score on the test set
    results['f_test'] = fbeta_score(y_test,predictions_test,beta=1)

    # Compute AUC on the the first 4000 training samples
    results['auc_train'] = roc_auc_score(y_train[:4000],predictions_train)

    # Compute AUC on the test set
    results['auc_test'] = roc_auc_score(y_test,predictions_test)

    # Success
    print "{} trained on {} samples.".format(learner.__class__.__name__, sample_size)
    print "{} with accuracy {}, F1 {} and AUC {}.".format(learner.__class__.__name__,\
          results['acc_test'],results['f_test'], results['auc_test'])
    # Return the results
    return results


import matplotlib.patches as mpatches
def evaluate(results,name):
    """
    Visualization code to display results of various learners.
```

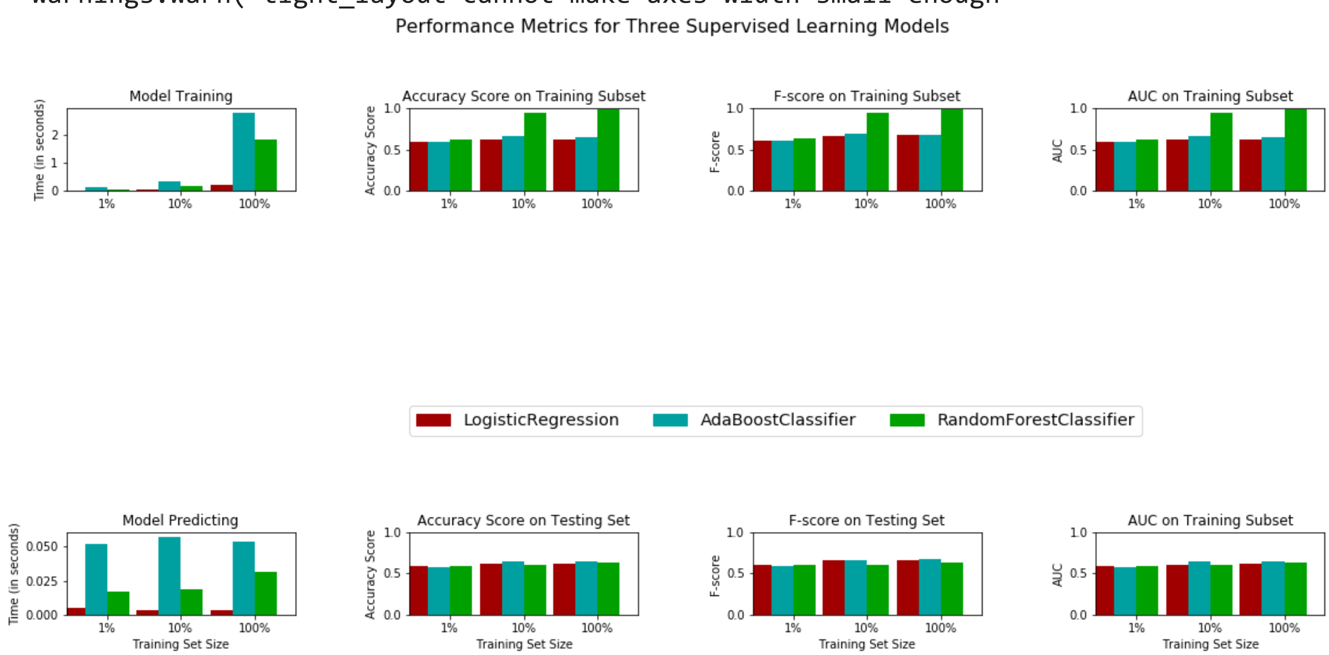
```
    inputs:
      - learners: a list of supervised learners
      - stats: a list of dictionaries of the statistic results from 'train_predict()'
      - accuracy: The score for the naive predictor
      - f1: The score for the naive predictor
    """

    # Create figure
    fig, ax = pl.subplots(2, 4, figsize = (16,7))

    # Constants
    bar_width = 0.3
    colors = ['#A00000','#00A0A0','#00A000']

    # Super loop to plot four panels of data
    for k, learner in enumerate(results.keys()):
        for j, metric in enumerate(['train_time', 'acc_train', 'f_train', 'auc_train','pred_t
                                    'f_test', 'auc_test']):
            for i in np.arange(3):

                # Creative plot code
                ax[j/4, j%4].bar(i+k*bar_width, results[learner][i][metric], width = bar_widt
                ax[j/4, j%4].set_xticks([0.45, 1.45, 2.45])
                ax[j/4, j%4].set_xticklabels(["1%", "10%", "100%"])
                ax[j/4, j%4].set_xlim((-0.1, 3.0))

    # Add labels
    ax[0, 0].set_ylabel("Time (in seconds)")
    ax[0, 1].set_ylabel("Accuracy Score")
    ax[0, 2].set_ylabel("F-score")
    ax[0, 3].set_ylabel("AUC")
    ax[1, 0].set_ylabel("Time (in seconds)")
    ax[1, 1].set_ylabel("Accuracy Score")
    ax[1, 2].set_ylabel("F-score")
    ax[1, 3].set_ylabel("AUC")
    ax[1, 0].set_xlabel("Training Set Size")
    ax[1, 1].set_xlabel("Training Set Size")
    ax[1, 2].set_xlabel("Training Set Size")
    ax[1, 3].set_xlabel("Training Set Size")

    # Add titles
    ax[0, 0].set_title("Model Training")
    ax[0, 1].set_title("Accuracy Score on Training Subset")
    ax[0, 2].set_title("F-score on Training Subset")
    ax[0, 3].set_title("AUC on Training Subset")
    ax[1, 0].set_title("Model Predicting")
    ax[1, 1].set_title("Accuracy Score on Testing Set")
    ax[1, 2].set_title("F-score on Testing Set")
    ax[1, 3].set_title("AUC on Training Subset")

    # Set y-limits for score panels
    ax[0, 1].set_ylim((0, 1))
```

```
    ax[0, 1].set_ylim((0, 1))
    ax[0, 2].set_ylim((0, 1))
    ax[0, 3].set_ylim((0, 1))
    ax[1, 1].set_ylim((0, 1))
    ax[1, 2].set_ylim((0, 1))
    ax[1, 3].set_ylim((0, 1))

    # Create patches for the legend
    patches = []
    for i, learner in enumerate(results.keys()):
        patches.append(mpatches.Patch(color = colors[i], label = learner))
    pl.legend(handles = patches,  bbox_to_anchor = (-1.4, 2.54),\
               loc = 'upper center', borderaxespad = 0., ncol = 3, fontsize = 'x-large')

    # Aesthetics
    pl.suptitle("Performance Metrics for Three Supervised Learning Models", fontsize = 16, y
    pl.savefig(name)
    pl.tight_layout()
    pl.show()
```

```
# Import the three supervised learning models from sklearn
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.linear_model import SGDClassifier
# Initialize the three models
clf_A = AdaBoostClassifier(random_state=0)
clf_B = LogisticRegression(random_state=0,C=1.0)
clf_C = RandomForestClassifier(random_state=0)

# Calculate the number of samples for 1%, 10%, and 100% of the training data
samples_1 = int(X_train_ADA.shape[0]*0.01)
samples_10 = int(X_train_ADA.shape[0]*0.1)
samples_100 = X_train_ADA.shape[0]

# Collect results on the learners
results = {}
for clf in [clf_A, clf_B, clf_C]:
    clf_name = clf.__class__.__name__
    results[clf_name] = {}
    for i, samples in enumerate([samples_1, samples_10, samples_100]):
        if clf == clf_A:
            results[clf_name][i] = \
            train_predict(clf, samples, X_train_ADA, y_train_ADA, X_test_ADA, y_test_ADA)
        elif clf == clf_B:
            results[clf_name][i] = \
```

```
            results[clf_name][i] = \
            train_predict(clf, samples, X_train_LR, y_train_LR, X_test_LR, y_test_LR)
        else:
            results[clf_name][i] = \
            train_predict(clf, samples, X_train_RF, y_train_RF, X_test_RF, y_test_RF)

# Run metrics visualization for the three supervised learning models chosen
evaluate(results,'perf_unopt.pdf')
```

```
AdaBoostClassifier trained on 356 samples.
AdaBoostClassifier with accuracy 0.582345523329, F1 0.586826347305 and AUC 0.5832947299
AdaBoostClassifier trained on 3567 samples.
AdaBoostClassifier with accuracy 0.644388398487, F1 0.669169404036 and AUC 0.6422416322
AdaBoostClassifier trained on 35679 samples.
AdaBoostClassifier with accuracy 0.654224464061, F1 0.683152299515 and AUC 0.6512698567
LogisticRegression trained on 356 samples.
LogisticRegression with accuracy 0.592433795712, F1 0.60469667319 and AUC 0.59238353800
LogisticRegression trained on 3567 samples.
LogisticRegression with accuracy 0.617402269861, F1 0.658871149089 and AUC 0.6128146071
LogisticRegression trained on 35679 samples.
LogisticRegression with accuracy 0.62421185372, F1 0.667855550602 and AUC 0.61914182571
RandomForestClassifier trained on 356 samples.
RandomForestClassifier with accuracy 0.594703656999, F1 0.606994375153 and AUC 0.594646
RandomForestClassifier trained on 3567 samples.
RandomForestClassifier with accuracy 0.607818411097, F1 0.606229425171 and AUC 0.609589
RandomForestClassifier trained on 35679 samples.
RandomForestClassifier with accuracy 0.631021437579, F1 0.634888944347 and AUC 0.632108
/usr/local/lib/python2.7/dist-packages/matplotlib/tight_layout.py:198: UserWarning: tig
  warnings.warn('tight_layout cannot make axes width small enough '
```



```
def gridsearch(clf,parameters,X_train, y_train, X_test, y_test):
    scorer = make_scorer(roc_auc_score)
    grid_obj = GridSearchCV(clf, parameters, scoring=scorer)

# Fit the grid search object to the training data and find the optimal parameters
    grid_fit = grid_obj.fit(X_train, y_train)

# Get the estimator
    best_clf = grid_fit.best_estimator_

# Make predictions using the unoptimized and model
    predictions = (clf.fit(X_train, y_train)).predict(X_test)
    best_predictions = best_clf.predict(X_test)

# Report the before-and-afterscores
    print (clf.__class__.__name__)
    print ("Unoptimized model\n------")
    print ("Accuracy score on testing data: {:.4f}".format(accuracy_score(y_test, predictions
    print ("F-score on testing data: {:.4f}".format(fbeta_score(y_test, predictions,beta=1))
    print ("AUC on testing data: {:.4f}".format(roc_auc_score(y_test, predictions)))
    print ("\nOptimized Model\n------")
    print ("Final accuracy score on the testing data: {:.4f}".format(accuracy_score(y_test, b
    print ("Final F-score on the testing data: {:.4f}".format(fbeta_score(y_test, best_predic
    print ("Final AUC on the testing data: {:.4f}".format(roc_auc_score(y_test, best_predicti

    print (best_clf)


# Do the grid search for hyperparameters
from sklearn.metrics import make_scorer
from sklearn.model_selection import GridSearchCV
parameters_RF = {"n_estimators": [10,20,50,100,250,500]}
parameters_LR = {"penalty": ['l1','l2'],
              "C": [0.1,0.5,1.,2.,2.5,5]}
parameters_ADA = {"n_estimators": [100,200,300,400],
              "learning_rate": [0.1,0.5,1]}


# Grid search for Adaboost
gridsearch(clf_A,parameters_ADA,X_train_ADA, y_train_ADA, X_test_ADA, y_test_ADA)
```

```
/usr/local/lib/python2.7/dist-packages/sklearn/model_selection/_split.py:2053: FutureWa
  warnings.warn(CV_WARNING, FutureWarning)
AdaBoostClassifier
Unoptimized model
------
Accuracy score on testing data: 0.6542
F-score on testing data: 0.6832
AUC on testing data: 0.6513

Optimized Model
------
Final accuracy score on the testing data: 0.6567
```

```
Final F-score on the testing data: 0.6873
Final AUC on the testing data: 0.6535
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=0.5, n_estimators=300, random_state=0)
```

```
# Grid search for logistic regression
gridsearch(clf_B,parameters_LR,X_train_LR, y_train_LR, X_test_LR, y_test_LR)
```

```
LogisticRegression
Unoptimized model
------
Accuracy score on testing data: 0.6242
F-score on testing data: 0.6679
AUC on testing data: 0.6191

Optimized Model
------
Final accuracy score on the testing data: 0.6247
Final F-score on the testing data: 0.6684
Final AUC on the testing data: 0.6196
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=0, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

```
# Grid search for RF
gridsearch(clf_C,parameters_RF,X_train_RF, y_train_RF, X_test_RF, y_test_RF)
```

```
RandomForestClassifier
Unoptimized model
------
Accuracy score on testing data: 0.6310
F-score on testing data: 0.6349
AUC on testing data: 0.6321

Optimized Model
------
Final accuracy score on the testing data: 0.6767
Final F-score on the testing data: 0.7074
Final AUC on the testing data: 0.6731
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', 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, n_estimators=500, n_jobs=None,
            oob_score=False, random_state=0, verbose=0, warm_start=False)
```

```
# Run the classifier with refined hyperparameters
clf_A = AdaBoostClassifier(random_state=0,learning_rate=0.5,n_estimators=300)
clf_B = LogisticRegression(random_state=0, C=2.5)
clf_C = RandomForestClassifier(random_state=0, n_estimators=500)

# Collect results on the learners
```

```
# Collect results on the learners
results = {}
for clf in [clf_A, clf_B, clf_C]:
    clf_name = clf.__class__.__name__
    results[clf_name] = {}
    for i, samples in enumerate([samples_1, samples_10, samples_100]):
        if clf == clf_A:
            results[clf_name][i] = \
            train_predict(clf, samples, X_train_ADA, y_train_ADA, X_test_ADA, y_test_ADA)
        elif clf == clf_B:
            results[clf_name][i] = \
            train_predict(clf, samples, X_train_LR, y_train_LR, X_test_LR, y_test_LR)
        else:
            results[clf_name][i] = \
            train_predict(clf, samples, X_train_RF, y_train_RF, X_test_RF, y_test_RF)

# Run metrics visualization for the three supervised learning models chosen
evaluate(results,'perf_opt.pdf')
```

```
AdaBoostClassifier trained on 356 samples.
AdaBoostClassifier with accuracy 0.576796973518, F1 0.58892699657 and AUC 0.57678333740
AdaBoostClassifier trained on 3567 samples.
AdaBoostClassifier with accuracy 0.639344262295, F1 0.663687676388 and AUC 0.6373126820
AdaBoostClassifier trained on 35679 samples.
AdaBoostClassifier with accuracy 0.656746532156, F1 0.687342062945 and AUC 0.6534651495
LogisticRegression trained on 356 samples.
```

```
# Run the classifier with diferent training/testing set split ratio
X_train_ADA, X_test_ADA, y_train_ADA, y_test_ADA = train_test_split(features_ADA, popular_lab

X_train_LR, X_test_LR, y_train_LR, y_test_LR = train_test_split(features_LR, popular_label, t

X_train_RF, X_test_RF, y_train_RF, y_test_RF = train_test_split(features_RF, popular_label, t

print "Training set has {} samples.".format(X_train_ADA.shape[0])
print "Testing set has {} samples.".format(X_test_ADA.shape[0])

samples_1 = int(X_train_ADA.shape[0]*0.01)
samples_10 = int(X_train_ADA.shape[0]*0.1)
samples_100 = X_train_ADA.shape[0]

clf_A = AdaBoostClassifier(random_state=0,learning_rate=0.5,n_estimators=300)
clf_B = LogisticRegression(random_state=0, C=2.5)
clf_C = RandomForestClassifier(random_state=0, n_estimators=500)

# Collect results on the learners
results = {}
for clf in [clf_A, clf_B, clf_C]:
    clf_name = clf.__class__.__name__
    results[clf_name] = {}
    for i, samples in enumerate([samples_1, samples_10, samples_100]):
        if clf == clf_A:
            results[clf_name][i] = \
            train_predict(clf, samples, X_train_ADA, y_train_ADA, X_test_ADA, y_test_ADA)
        elif clf == clf_B:
            results[clf_name][i] = \
            train_predict(clf, samples, X_train_LR, y_train_LR, X_test_LR, y_test_LR)
        else:
            results[clf_name][i] = \
            train_predict(clf, samples, X_train_RF, y_train_RF, X_test_RF, y_test_RF)

# Run metrics visualization for the three supervised learning models chosen
evaluate(results,'perf_opt_test.pdf')
```

```
Training set has 33697 samples.
Testing set has 5947 samples.
AdaBoostClassifier trained on 336 samples.
AdaBoostClassifier with accuracy 0.57642508828, F1 0.609033059134 and AUC 0.57358017246
AdaBoostClassifier trained on 3369 samples.
AdaBoostClassifier with accuracy 0.632924163444, F1 0.667883766925 and AUC 0.6290502894
AdaBoostClassifier trained on 33697 samples.
AdaBoostClassifier with accuracy 0.649403060367, F1 0.681825118266 and AUC 0.6457628875
LogisticRegression trained on 336 samples.
LogisticRegression with accuracy 0.587691272911, F1 0.578983516484 and AUC 0.5905853870
LogisticRegression trained on 3369 samples.
LogisticRegression with accuracy 0.620480914747, F1 0.661568451042 and AUC 0.6156849412
LogisticRegression trained on 33697 samples.
LogisticRegression with accuracy 0.624348410964, F1 0.669037037037 and AUC 0.6188174509
RandomForestClassifier trained on 336 samples.
RandomForestClassifier with accuracy 0.601479737683, F1 0.644464446445 and AUC 0.596654
RandomForestClassifier trained on 3369 samples.
RandomForestClassifier with accuracy 0.646880780225, F1 0.685157421289 and AUC 0.642154
RandomForestClassifier trained on 33697 samples.
RandomForestClassifier with accuracy 0.674794013788, F1 0.705900243309 and AUC 0.671017
```

Performance Metrics for Three Supervised Learning Models