Main.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 22 17:24:53 2018

@author: xin

"""

from sklearn.model_selection import train_test_split
import MyMethods as myFunc

X, y = myFunc.importRawDataCleaning()

# Split the dataset in two parts : train 70%, test 30%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
train_class = y_train.value_counts()
test_class = y_test.value_counts()
print("class distribution for balance: train: \n"+str(train_class)+"test: \n"+str(test_class))

# standarlize training data and test data
X_train, X_test = myFunc.dataStandarlize(X_train, X_test)
y_train.index = range(len(y_train)) #reset index from 0 to match train data index
y_test.index = range(len(y_test))


# feature selection
num_class = y_train.value_counts
X_train, X_test = myFunc.featureSel(X_train, y_train, X_test, num_class)


# parameter estimation
myFunc.paramEstimateSVM(X_train, X_test, y_train, y_test)
myFunc.paramEstimateRF(X_train, y_train)
myFunc.paramEstimateKNN(X_train, y_train)

# predict test set and get AUC with cross_validation to compare various models
myFunc.TrainClassification(X_train, X_test, y_train, y_test)
```

MyMethods.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Apr 10 12:35:09 2018

@author: xin
"""
from __future__ import print_function
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.manifold import TSNE

from sklearn.learning_curve import validation_curve
from sklearn.model_selection import GridSearchCV
from sklearn import model_selection
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold

from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.linear_model import Perceptron
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import classification_report

import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt

def importRawDataCleaning():
    # load data
    df = pd.read_csv('OnlineNewsPopularityReduced.csv')
```

```python
# Juding if there have categerical features except for url, which is just like sample's index instead of features
isChar = df.iloc[:,2:df.columns.size].select_dtypes(include=['object']).dtypes
if isChar.empty :
    print ('in the data set, there are all numerical.')
else:
    print ('#need process them to numerical features with considering unorder or order etc.')

# missing data, drop the whole related row of a sample if there have any features missing
df.dropna(axis = 0, how = 'any')

# drop duplicated data
df.drop_duplicates()

# 1 - slice 'n_tokens_title' to 'abs_title_sentiment_polarity' as features
features = df.iloc[:,2:(df.columns.size-1)]
# 2 - remove features that all values are the same like "kw_min_min"
features = features.loc[:, (features != features.loc[0]).any()]

# split features and labels
# multi-class, range and make the number of each class balanced as much as possible
def func(x):
    num_class = 5
    if x <= 900:
        return 1
    elif x <= 1200:
        return 2
    elif x <= 1600:
        return 3
    elif x <= 3400:
        return 4
    else:
        return 5

# two-class
def func2(x):
    num_class = 2
    if x <= 1300:
        return 0
    else:
```

```
        return 1
    label = df['shares'].map(func2)
    balanced = label.value_counts()
    print("balanced : \n"+str(balanced))
    return features, label

# data standarlize - fit by train data, and then also standarlize test data
def dataStandarlize(X_train, X_test):
    # abnormal data - remove outlier through standardize data by robustScaler
    # centering and scaling data
    #scaler = StandardScaler()
    scaler = RobustScaler(quantile_range=(0.01, 99.99)) #initializes a StandardScaler object
    scaler.fit(X_train)
    X_train = pd.DataFrame(scaler.transform(X_train))#, index=X_train.index)
    X_test = pd.DataFrame(scaler.transform(X_test))#, index=X_test.index)

    return X_train, X_test

def featureSel(X_train, y_train, X_test, num_class):

    num_features = X_train.shape[1]

    # feature selection dimensional reduction using PCA
    # since a downstream model can further make some assumption on the linear independence of the features, so use PCA
    # with whiten=True to further remove the linear correlation across features.

    # choose target dimension using scree plot
    U, S, V = np.linalg.svd(X_train)
    eigvals = S**2 / np.cumsum(S)[-1]
    #fig = plt.figure(figsize=(8,5))
    sing_vals = np.arange(num_features) + 1

    plt.plot(sing_vals, eigvals, 'ro-', linewidth=2)
    plt.title('Scree Plot')
    plt.xlabel('Principal Component')
    plt.ylabel('Eigenvalue')

    leg = plt.legend(['Eigenvalues from SVD'], loc='best', borderpad=0.3, shadow=False,
                     prop=matplotlib.font_manager.FontProperties(size='small'), markerscale=0.4)
    leg.get_frame().set_alpha(0.4)
```

```
    leg.draggable(state=True)
    plt.show()

    #only select singular value > 1 features, based on S of svd and variance_ratio sum=1
    num_features = np.sum(S > 1)

    pca = PCA(n_components=num_features, whiten=True)
    pca.fit(X_train.values)
    X_train = pd.DataFrame(pca.transform(X_train.values))
    X_test = pd.DataFrame(pca.transform(X_test.values))

    print("feature numbers: "+str(num_features))
    print("explained_variance_ratio_: \n"+str(pca.explained_variance_ratio_))
    print("explained_variance_ratio_.cumsum: \n"+str(pca.explained_variance_ratio_.cumsum()))


    lda = LDA(n_components=num_features)
    lda.fit(X_train.values, y_train)

    #t-SNE better for higher scatter different classes, but only accept number of features is inferior to 4
    #tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=250)
    #new3 = tsne.fit_transform(features.values)
    #features3 = pd.DataFrame(new3)

    #print("explained_variance_ratio_"+str(tsne.explained_variance_ratio_))
    #print("explained_variance_ratio_.cumsum"+str(tsne.explained_variance_ratio_.cumsum()))

    #plt.scatter(features[label==1][0], features[label==1][1], label='Class 1', c='red')
    #plt.scatter(features[label==2][0], features[label==2][1], label='Class 2', c='blue')
    #if num_class == 5:
    #    plt.scatter(features[label==3][0], features[label==3][1], label='Class 3', c='lightgreen')
    #    plt.scatter(features[label==4][0], features[label==4][1], label='Class 4', c='purple')
    #    plt.scatter(features[label==5][0], features[label==5][1], label='Class 5', c='yellow')

    #plt.legend()
    #plt.show()

    return X_train, X_test

def TrainClassification(X_train, X_test, y_train, y_test):
```

```
    # prepare configuration for cross validation test harness
    seed = 7
    # prepare models
    models = []
    models.append(('KNN', KNeighborsClassifier(20)))
    models.append(('Linear SVM', SVC(kernel="linear", C=100)))
    models.append(('RBF SVM', SVC(gamma=1e-2, C=100)))
    models.append(('Perceptron', Perceptron()))
    models.append(('RandForest', RandomForestClassifier(max_depth=5, n_estimators=500, max_features=1)))
#n_estimators=10
    models.append(('NaiveBayes', GaussianNB()))

    # evaluate each model in turn
    results = []
    names = []
    scoring = 'accuracy'
    for name, model in models:
        kfold = model_selection.KFold(n_splits=5, random_state=seed)
        cv_results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold, scoring=scoring)
        results.append(cv_results)
        names.append(name)

        model.fit(X_train, y_train)
        roc_auc = roc_auc_score(y_test, model.predict(X_test))
        if name == 'Linear SVM':
            print("support_vectors: "+str(model.support_vectors_))
            print("support_vectors dim: row:"+str(len(model.support_vectors_))+"\r col:"+str(len(model.support_vectors_[0])))

        print("Detailed classification report:")
        print()
        print("The model is trained on the full development set.")
        print("The scores are computed on the full evaluation set.")
        msg = "%s: AUC = %2.2f" % (name, roc_auc)
        print(msg)
        print(classification_report(y_test, model.predict(X_test)))

    # boxplot algorithm comparison
    fig = plt.figure()
    fig.suptitle('Algorithm Comparison')
    ax = fig.add_subplot(111)
```

```
    plt.boxplot(results)
    ax.set_xticklabels(names)
    plt.show()

    return


def paramEstimateSVM(X_train, X_test, y_train, y_test):
    scores = ['precision', 'recall']

    param_range = np.logspace(-4, -1, 8)

    train_scores, test_scores = validation_curve(
    SVC(), X_train, y_train, param_name="gamma", param_range=param_range,
    cv=5, scoring="accuracy", n_jobs=1)

    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.title("Validation Curve with SVM - param gamma")
    plt.xlabel("gamma")
    plt.ylabel("Score")
    plt.ylim(0.4, 1.0)
    lw = 2
    plt.semilogx(param_range, train_scores_mean, label="Training score",
              color="darkorange", lw=lw)
    plt.fill_between(param_range, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.2,
                 color="darkorange", lw=lw)
    plt.semilogx(param_range, test_scores_mean, label="Cross-validation score",
              color="r", lw=lw)
    plt.fill_between(param_range, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.2,
                 color="g", lw=lw)
    plt.legend(loc="best")
    plt.show()

    # Set the parameters by cross-validation
```

```
    tuned_parameters = [{'kernel': ['linear'], 'C': [1, 10, 100, 500, 1000]},
                        {'kernel': ['rbf'], 'gamma': [1e-4, 6e-3, 1e-2],'C': [1, 10, 100, 500, 1000]}]
    scores = ['precision', 'recall']
    for score in scores:
        print("# Tuning hyper-parameters for %s" % score)
        print()

        clf = GridSearchCV(SVC(), tuned_parameters, cv=5, scoring='%s_macro' % score)
        clf.fit(X_train, y_train)
        print("support_vectors: "+clf.support_vectors_)
        print("SVM Best parameters set found on development set:")
        print()
        print(clf.best_params_)
        print()
        print("Grid scores on development set:")
        print()
        means = clf.cv_results_['mean_test_score']
        stds = clf.cv_results_['std_test_score']
        for mean, std, params in zip(means, stds, clf.cv_results_['params']):
            print("%0.3f (+/-%0.03f) for %r"
                  % (mean, std * 2, params))
        print()
    return

def paramEstimateRF(X_train, y_train):
    # Set the parameters by cross-validation
    tuned_parameters = [{'n_estimators': [50, 100, 200, 300, 400, 500]}]
    scores = ['precision', 'recall']
    for score in scores:
        print("# Tuning hyper-parameters for %s" % score)
        print()

        clf = GridSearchCV(RandomForestClassifier(), tuned_parameters, cv=5 ,scoring='%s_macro' % score)
        clf.fit(X_train, y_train)

        print("RF Best parameters set found on development set:")
        print()
        print(clf.best_params_)
        print()
```

```
        print("Grid scores on development set:")
        print()
        means = clf.cv_results_['mean_test_score']
        stds = clf.cv_results_['std_test_score']
        for mean, std, params in zip(means, stds, clf.cv_results_['params']):
            print("%0.3f (+/-%0.03f) for %r"
                  % (mean, std * 2, params))
    return

def paramEstimateKNN(X_train, y_train):
    # Set the parameters by cross-validation
    tuned_parameters = [{'n_neighbors': [2, 3, 5, 6, 8, 10, 20, 30, 40, 50]}]
    scores = ['precision', 'recall']
    for score in scores:
        print("# Tuning hyper-parameters for %s" % score)
        print()

        clf = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5 ,scoring='%s_macro' % score)
        clf.fit(X_train, y_train)

        print("KNN Best parameters set found on development set:")
        print()
        print(clf.best_params_)
        print()
        print("Grid scores on development set:")
        print()
        means = clf.cv_results_['mean_test_score']
        stds = clf.cv_results_['std_test_score']
        for mean, std, params in zip(means, stds, clf.cv_results_['params']):
            print("%0.3f (+/-%0.03f) for %r"
                  % (mean, std * 2, params))
    return
```