POC_FindStance_V2

January 5, 2018

```
In [1]: import math as m
        import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        from sklearn import svm
        from sklearn import preprocessing
        from sklearn.svm import LinearSVC
        from sklearn.preprocessing import LabelEncoder
        from sklearn.model_selection import train_test_split
        from sklearn.naive_bayes import GaussianNB
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.tree import DecisionTreeClassifier
        import warnings
        warnings.filterwarnings('ignore')

        np.set_printoptions(threshold=np.inf)
```

1 Chargement des données

```
In [2]: df_train = pd.read_csv('./dataset/train_ingrid.csv')
        df_test = pd.read_csv('./dataset/test_ingrid.csv')
```

1.1 Fusion des données

```
In [3]: target = np.hstack([np.array(df_train['Target']), np.array(df_test['Target'])])
        opinion = np.hstack([np.array(df_train['Opinion Towards']), np.array(df_test['Opinion To
        sentiment = np.hstack([np.array(df_train['Sentiment']), np.array(df_test['Sentiment'])])
        stance = np.hstack([np.array(df_train['Stance']), np.array(df_test['Stance'])])
```

1.2 Encodage des labels (passage en classes)

```
In [4]: lb_target = preprocessing.LabelEncoder()
        target_label = lb_target.fit_transform(target)
        lb_opinion = preprocessing.LabelEncoder()
        opinion_label = lb_opinion.fit_transform(opinion)
        lb_sentiment = preprocessing.LabelEncoder()
        sentiment_label = lb_sentiment.fit_transform(sentiment)
```

```
lb_stance = preprocessing.LabelEncoder()
stance_label = lb_stance.fit_transform(stance)
```

1.3 Découpage en ensemble d'apprentissage et de test

```
In [5]: X = np.array([target_label, opinion_label, sentiment_label])
        X = np.transpose(X)
        X_train, X_test, y_train, y_test= train_test_split( X, stance_label, test_size=0.15, ran
```

2 Tests de différentes méthodes

2.1 Bayes

```
In [6]: model = GaussianNB()
        # Train the model using the training sets
        model.fit(X_train, y_train)
        #Predict Output
        predicted = model.predict(X_test)
        erreur = sum((y_test - predicted) != 0)/len(y_test)*100
        erreur

Out[6]: 31.737346101231189
```

2.2 Linear SVM

```
In [7]: clf = LinearSVC(random_state=0)
        clf.fit(X_train, y_train)
        dec = clf.predict(X_test)
        erreur = sum((y_test - dec) != 0)/len(y_test)*100
        erreur

Out[7]: 37.61969904240766
```

2.3 SVM

```
In [8]: clf = svm.SVC(C=100,gamma=0.001)
        clf.fit(X_train, y_train)
        dec = clf.predict(X_test)
        erreur = sum((y_test - dec) != 0)/len(y_test)*100
        erreur

Out[8]: 33.105335157318741
```

2.4 K plus proches voisins

```
In [9]: model = KNeighborsClassifier(4)
        # Train the model using the training sets
        model.fit(X_train, y_train)
        #Predict Output
```

```
predicted = model.predict(X_test)
erreur = sum((y_test - predicted) != 0)/len(y_test)*100
erreur
```

```
Out[9]: 34.473324213406293
```

2.5 Arbre de décision

```
In [10]: model = DecisionTreeClassifier(max_depth=5)
         # Train the model using the training sets
         model.fit(X_train, y_train)
         #Predict Output
         predicted = model.predict(X_test)
         erreur = sum((y_test - predicted) != 0)/len(y_test)*100
         erreur
```

```
Out[10]: 24.48700410396717
```

3 Test paramètres SVM

```
In [21]: clf = svm.SVC(kernel='linear', C=100,gamma=0.001)
         clf.fit(X_train, y_train)
         dec = clf.predict(X_test)
         erreur = sum((y_test - dec) != 0)/len(y_test)*100
         erreur
```

```
Out[21]: 34.610123119015043
```

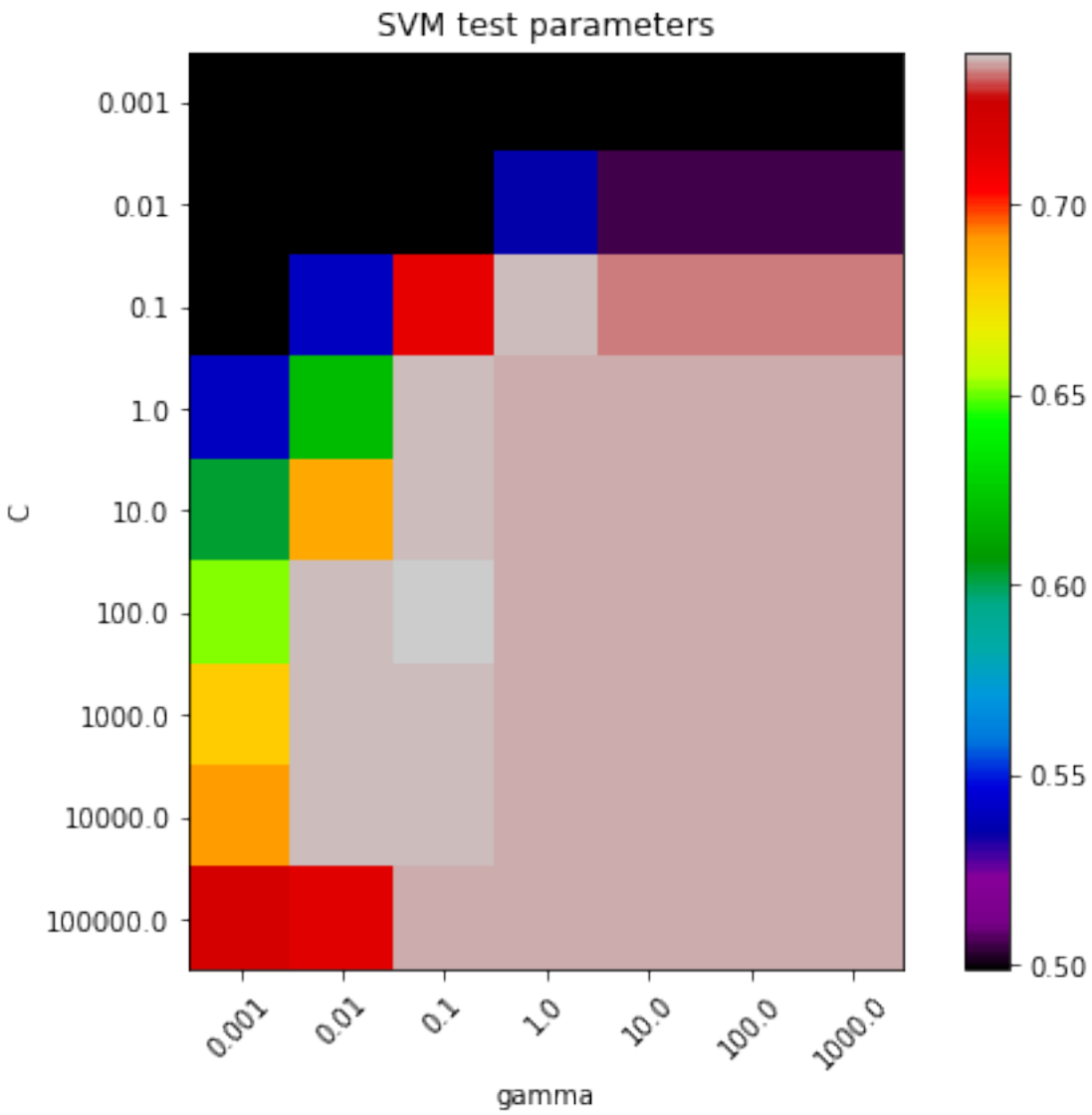
```
In [22]: clf = svm.SVC(kernel='rbf', C=100,gamma=0.001)
         clf.fit(X_train, y_train)
         dec = clf.predict(X_test)
         erreur = sum((y_test - dec) != 0)/len(y_test)*100
         erreur
```

```
Out[22]: 33.105335157318741
```

```
In [27]: from sklearn.grid_search import GridSearchCV
         from sklearn.cross_validation import StratifiedKFold

         C_range = 10. ** np.arange(-3, 6)
         gamma_range = 10. ** np.arange(-5, 4)
         param_grid = dict(gamma=gamma_range, C=C_range)
         grid = GridSearchCV(svm.SVC(), param_grid=param_grid, cv=StratifiedKFold(y=y_train))
         grid.fit(X_train, y_train)

         print("The best classifier is: ", grid.best_estimator_)
```

```
The best classifier is:  SVC(C=100.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.10000000000000001,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

```
In [26]: import pylab as pl
         # plot the scores of the grid
         # grid_scores_ contains parameter settings and scores
         score_dict = grid.grid_scores_

         # We extract just the scores
         scores = [x[1] for x in score_dict]
         scores = np.array(scores).reshape(len(C_range), len(gamma_range))

         # Make a nice figure
         pl.figure(figsize=(8, 6))
         pl.subplots_adjust(left=0.15, right=0.95, bottom=0.15, top=0.95)
         pl.imshow(scores, interpolation='nearest', cmap=pl.cm.spectral)
         pl.xlabel('gamma')
         pl.ylabel('C')
         pl.title('SVM test parameters')
         pl.colorbar()
         pl.xticks(np.arange(len(gamma_range)), gamma_range, rotation=45)
         pl.yticks(np.arange(len(C_range)), C_range)
         pl.show()
```




```
In [23]: clf = svm.SVC(kernel='rbf', C=100,gamma=0.1)
         clf.fit(X_train, y_train)
         dec = clf.predict(X_test)
         erreur = sum((y_test - dec) != 0)/len(y_test)*100
         erreur

Out[23]: 24.48700410396717
```