

Rapport Python for Data Analysis

MARC GUYNOT ESILV-S9

2020



Objectif

A partir d'un dataset de 11k observations de 561 variables issues de différents capteurs d'un téléphone accroché à une ceinture sur 30 individus différents.

> Déterminer l'une des 12 postures labellisées

Process

Pipeline data



Intégration DataSet			
		DataVisualisation	DataVisualisation
		Feature engineering	Feature engineering
Models préparation			
	Models tuning		Models tuning
Intégration Django			
1) Préparation environnement & 1 ^{ère} prédiction	2) Models tuning	3) Data visualisation et affinage prédiction	Itérations sur affinage prédiction

Développement

1) Préparation environnement & 1^{ère} prédiction

```
1 labels = pd.read_csv("https://raw.githubusercontent.com/Koalanas/pythonfordata/master/datas/features.txt", header=None)
2 l = labels[0].tolist()
3 labels = []
4 i = 0
5 for la in l:
6     i += 1
7     la = la.replace(" ", "")
8     while(la in labels):
9         la = la+"x"
10    labels.append(la)
11
12 train_x = pd.read_csv("https://raw.githubusercontent.com/Koalanas/pythonfordata/master/datas/Train/X_train.txt", " ", header=None, names=labels)
13 train_y = pd.read_csv("https://raw.githubusercontent.com/Koalanas/pythonfordata/master/datas/Train/y_train.txt", " ", header=None, names=['y_target'])
14 train = pd.concat([train_x, train_y], axis=1)
15
16 test_x = pd.read_csv("https://raw.githubusercontent.com/Koalanas/pythonfordata/master/datas/Test/X_test.txt", " ", header=None, names=labels)
17 test_y = pd.read_csv("https://raw.githubusercontent.com/Koalanas/pythonfordata/master/datas/Test/y_test.txt", " ", header=None, names=['y_target'])
18 test = pd.concat([test_x, test_y], axis=1)
19
20 df = pd.concat([train, test])
21 df.shape
```

(10929, 562)

➤ Import des données et création du train-test

11k observations et 561 variables

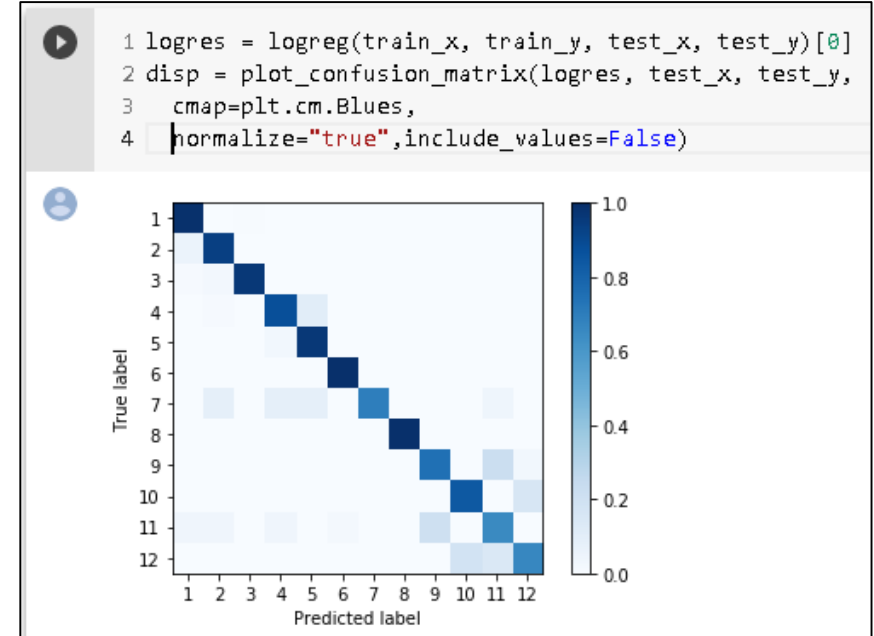
1) Préparation environnement & 1^{ère} prédiction

```

1 def logreg(trainx, trainy, testx, testy):
2     from sklearn.linear_model import LogisticRegression
3     time_start = time.clock()
4     logres = LogisticRegression(solver='lbfgs', multi_class="ovr", max_iter=1000)
5     logres.fit(trainx, trainy.values.ravel())
6     predictions = logres.predict(testx)
7     i = 0
8     accuracy = 0
9     for res in predictions:
10         if res == testy.loc[i, "y_target"]:
11             accuracy += 1
12         i += 1
13     accuracy = (accuracy/i)
14     time_elapsed = (time.clock() - time_start)
15     return [logres, accuracy, time_elapsed]
16
17 logreg(train_x, train_y, test_x, test_y)

```

[LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, l1_ratio=None, max_iter=1000, multi_class='ovr', n_jobs=None, penalty='l2', random_state=None, solver='lbfgs', tol=0.0001, verbose=0, warm_start=False), 0.9468690702087287, 76.86743899999999]



➤ Premier modèle de test non paramétré

Régression logistique avec 94,7 % de précision pour un temps d'exécution de 77 secondes

1) Préparation environnement & 1^{ère} prédiction

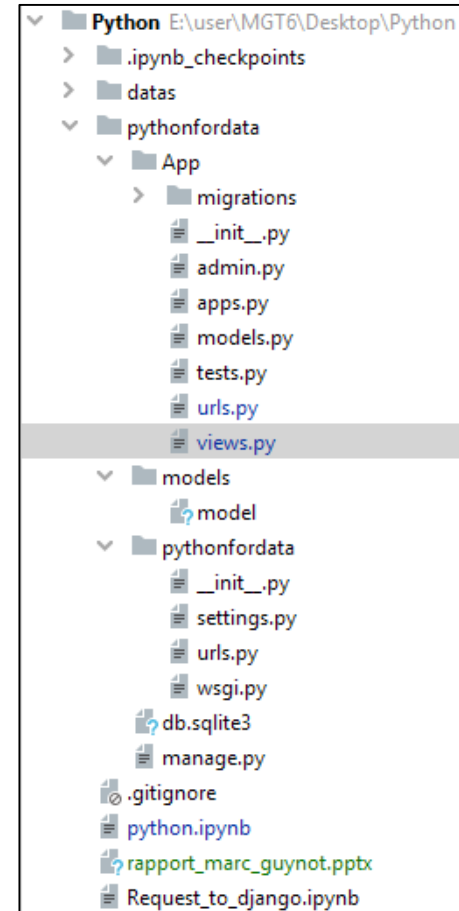
```
1 modellog = logreg(train_x, train_y, test_x, test_y)[0]
2 with open("pythonfordata/models/model", 'wb') as file:
3     pickle.dump(modellog, file)
```

1) Export du model

```
class Predict.views.APIView:
    def post(self, request):
        result = 0
        self.request.POST._mutable = True
        rpd = pd.read_json(request.data.pop('demande')[0])
        print("Requette de " + str(rpd.shape[0]) + " observations")
        model_name = "model"
        path = os.path.join(settings.MODEL_ROOT, model_name)
        with open(path, 'rb') as file:
            model = pickle.load(file)
        try:
            result = model.predict(rpd)
        except Exception as err:
            return Response(str(err), status=status.HTTP_400_BAD_REQUEST)

        return Response(result, status=status.HTTP_200_OK)
```

3) views.py : Class Predict



2) Architecture Django

1) Préparation environnement & 1^{ère} prédiction

```
Entrée [157]: def request(df):
               requetteJson = json.loads(df.to_json())
               URL = "http://127.0.0.1:8000/App/predict/"
               results = json.loads(requests.post(url=URL, data = {'demande':json.dumps(requetteJson)}).text)
               tmp = []
               for r in results:
                   tmp.append(y_label[r-1])
               results = tmp
               return results

               time_start = time.time()
               testme = (test_x.sample(frac=1))[:5]
               print(time.time() - time_start)
               request(testme)

0.01737356185913086

Out[157]: ['WALKING_DOWNSTAIRS',
           'LAYING',
           'WALKING',
           'WALKING_UPSTAIRS',
           'WALKING_UPSTAIRS']
```

```
Django version 2.1, using settings 'pythonfordata.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
Requette de 5 observations
[31/Jan/2020 13:52:37] "POST /App/predict/ HTTP/1.1" 200 11
Requette de 10 observations
[31/Jan/2020 13:52:46] "POST /App/predict/ HTTP/1.1" 200 21
Requette de 1 observations
[31/Jan/2020 13:52:49] "POST /App/predict/ HTTP/1.1" 200 3
Requette de 1000 observations
[31/Jan/2020 13:52:58] "POST /App/predict/ HTTP/1.1" 200 2036
```

Logs Django

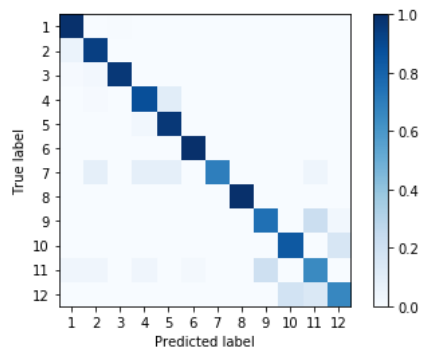
Test de l'intégration sur le serveur local Django

- Envoie de 5 observations
- Réponse en 18ms (serveur local), quelque soit le nombre d'observation envoyées

2) Models tuning

```
1 def logreg(trainx, trainy, testx, testy):
2     from sklearn.linear_model import LogisticRegression
3     time_start = time.clock()
4     logres = LogisticRegression(solver='lbfgs', multi_class="ovr",
5                               max_iter=1000)
6     logres.fit(trainx, trainy.values.ravel())
7     predictions = logres.predict(testx)
8     i = 0
9     accuracy = 0
10    for res in predictions:
11        if res == testy.loc[i, "y_target"]:
12            accuracy += 1
13        i += 1
14    accuracy = (accuracy/i)
15    time_elapsed = (time.clock() - time_start)
16    confusion_matrix(testy, predictions)
17    return [logres, accuracy, time_elapsed]
18
19 logres = logreg(train_x, train_y, test_x, test_y)
20 print("Précision : " + str(logres[1]) + ", temps : " + str(logres[2]) )
21 disp = plot_confusion_matrix(logres[0], test_x, test_y,
22                               cmap=plt.cm.Blues,
23                               normalize="true", include_values=False)
```

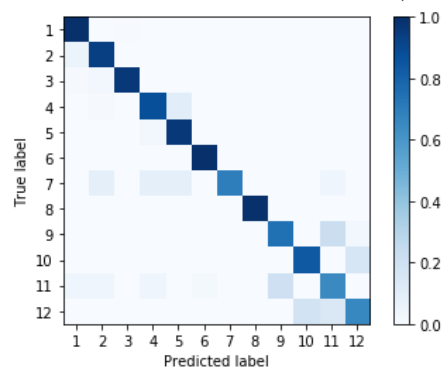
Précision : 0.9468690702087287, temps : 75.13321099999985



Regression logistique

```
1 def rfc(trainx, trainy, testx, testy):
2     from sklearn.ensemble import RandomForestClassifier
3     time_start = time.clock()
4     rfc = RandomForestClassifier(n_estimators = 200)
5     rfc.fit(trainx, trainy.values.ravel())
6     predictions = rfc.predict(testx)
7     i = 0
8     accuracy = 0
9     for res in predictions:
10        if res == testy.loc[i, "y_target"]:
11            accuracy += 1
12        i += 1
13    accuracy = (accuracy/i)
14    time_elapsed = (time.clock() - time_start)
15    return [logres, accuracy, time_elapsed]
16
17 rf = rfc(train_x, train_y, test_x, test_y)
18 print("Précision : " + str(rf[1]) + ", temps : " + str(rf[2]) )
19 disp = plot_confusion_matrix(rf[0], test_x, test_y,
20                               cmap=plt.cm.Blues,
21                               normalize="true", include_values=False)
```

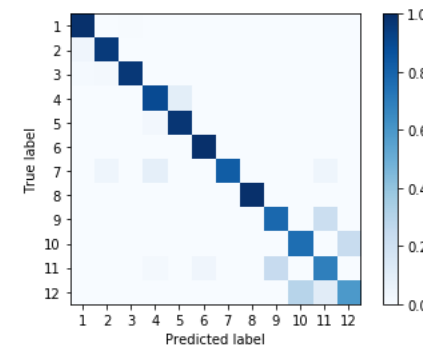
Précision : 0.9092346616065781, temps : 33.36913100000015



RandomForest Classifier

```
1 def svm(trainx, trainy, testx, testy):
2     from sklearn import svm
3     time_start = time.clock()
4     clf = svm.SVC(gamma=0.001, C=100., decision_function_shape='ovr',
5                  random_state=22)
6     clf.fit(trainx, trainy.values.ravel())
7     predictions = clf.predict(testx)
8     i = 0
9     accuracy = 0
10    for res in predictions:
11        if res == testy.loc[i, "y_target"]:
12            accuracy += 1
13        i += 1
14    accuracy = (accuracy/i)
15    time_elapsed = (time.clock() - time_start)
16    return [clf, accuracy, time_elapsed]
17
18 svm = svm(train_x, train_y, test_x, test_y)[1]
19 sv = svm(train_x, train_y, test_x, test_y)
20 print("Précision : " + str(sv[1]) + ", temps : " + str(sv[2]) )
21 disp = plot_confusion_matrix(sv[0], test_x, test_y,
22                               cmap=plt.cm.Blues,
23                               normalize="true", include_values=False)
```

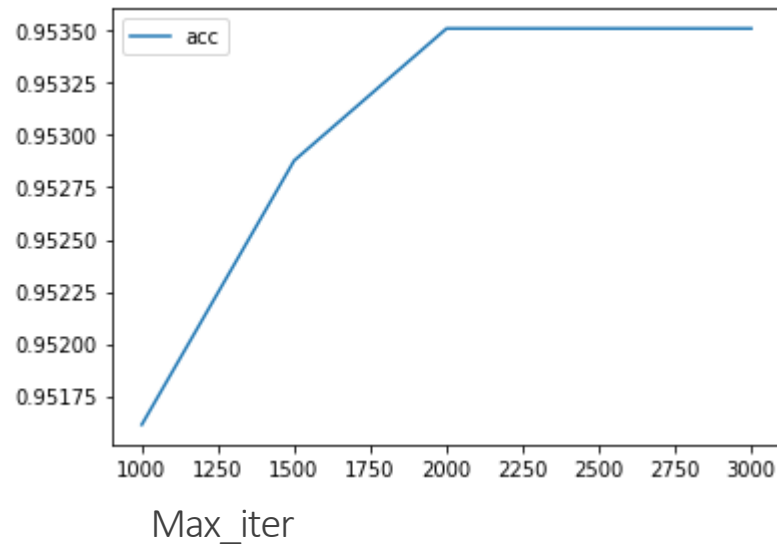
Précision : 0.9535104364326376, temps : 9.435485999999855



Support Vector Machines

2) Models tuning

Support Vector Machines Tunning



C [0.1, 1, 10, 100, 1000]

acc [0.7504743833017078, 0.8981657179000633, 0.9471853257432005, 0.9535104364326376, 0.952561669829222]

Regularisation parameter

tol : [0.1, 0.01, 0.001, 0.0001, 1e-05, 1e-08]

acc : [0.9529, 0.9535, 0.9535, 0.9538, 0.9538, 0.9538]

Tolerance parameter

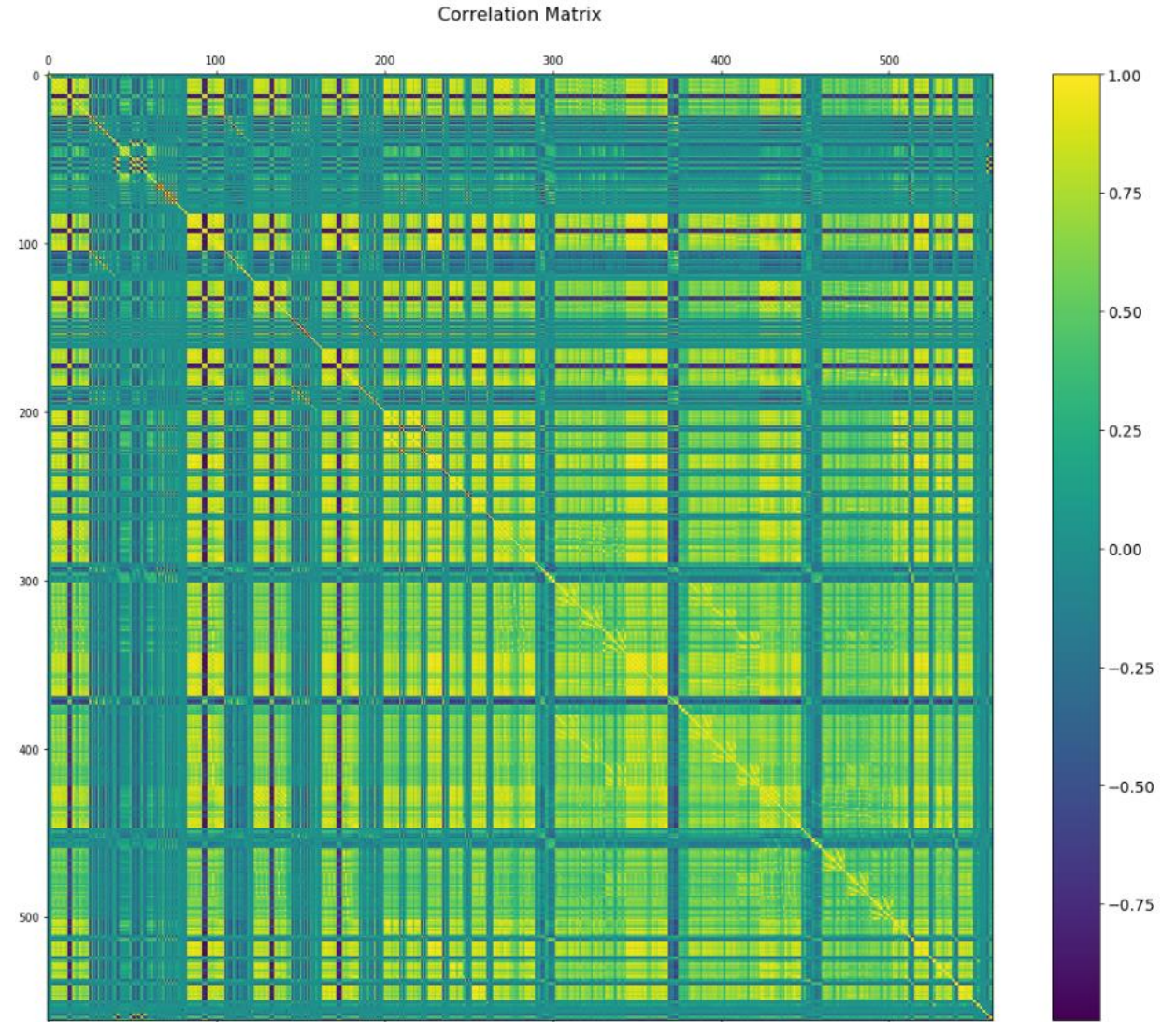
Paramètres retenus :

```
svm.SVC(gamma=0.001, C=100., decision_function_shape='ovr', random_state=22, max_iter=2500, kernel='rbf', tol=0.0001)
```

3) Data visualisation et affinage prédiction

Features engineering :

- La data est déjà très propre, aucun NA pas de données hors normes
- Seulement des valeurs numériques
- Pas de création de variables imaginable, les données sont certes brutes mais trop abstraites
- Le meilleurs traitement à faire serait de retirer des variables car il y en a au total 561



3) Data visualisation et affinage prédiction

12 lines (12 sloc) | 255 Bytes

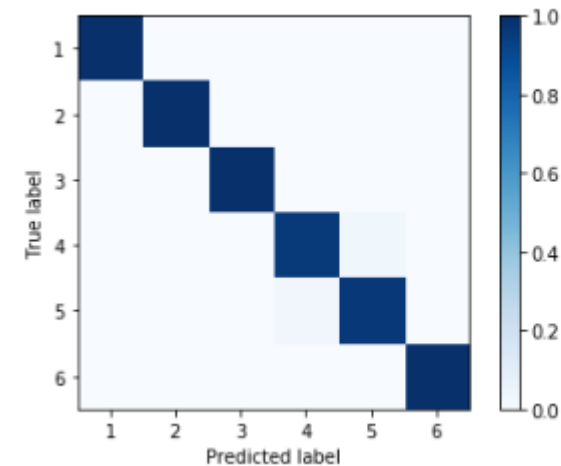
```
1 1 WALKING
2 2 WALKING_UPSTAIRS
3 3 WALKING_DOWNSTAIRS
4 4 SITTING
5 5 STANDING
6 6 LAYING
7 7 STAND_TO_SIT
8 8 SIT_TO_STAND
9 9 SIT_TO_LIE
10 10 LIE_TO_SIT
11 11 STAND_TO_LIE
12 12 LIE_TO_STAND
```

Features engineering :

- En enlevant les postures intermédiaires qui sont présentes en sous nombre on améliore la prédiction

```
1 dfLight = df[train['y_target'] < 7]
2 dfLight_x = trainLight.iloc[ : ,-1]
3 dfLight_y = trainLight.iloc[ : ,:-1]
4
5 trainLight_y, testLight_y, trainLight_x, testLight_x =
6 train_test_split(dfLight_x, dfLight_y, test_size=0.30, random_state=42)
```

Précision : 0.9865168539325843, temps : 3.8325029999999633



Modèle SVC : 98.7 % de précision