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 tunning		Models tunning
Intégration Django			
1) Préparation environnement & 1ère prédiction	2) Models tunning	3) Data visualisation et affinage prédiction	Itérations sur affinage prédiction

Développement

1) Préparation environnement & 1ère prédiction

```
labels = pd.read_csv("https://raw.githubusercontent.com/Koalananas/pythonfordata/master/datas/features.txt", header=None)
l = labels[0].tolist()
labels = []
i = 0
for la in l:
  i += 1
  la = la.replace(" ", "")
  while(la in labels):
    la = la+"x"
  labels.append(la)

train_x = pd.read_csv("https://raw.githubusercontent.com/Koalananas/pythonfordata/master/datas/Train/X_train.txt"," ", header=None, names=labels)
train_y = pd.read_csv("https://raw.githubusercontent.com/Koalananas/pythonfordata/master/datas/Train/y_train.txt"," ", header=None, names=['y_target'])
train  = pd.concat([train_x, train_y], axis=1)

test_x = pd.read_csv("https://raw.githubusercontent.com/Koalananas/pythonfordata/master/datas/Test/X_test.txt"," ", header=None, names=labels)
test_y = pd.read_csv("https://raw.githubusercontent.com/Koalananas/pythonfordata/master/datas/Test/y_test.txt"," ", header=None, names=['y_target'])
test  = pd.concat([test_x, test_y], axis=1)

df = pd.concat([train, test])
df.shape
```

```
(10929, 562)
```

- Import des données et création du train-test

11k observations et 561 variables

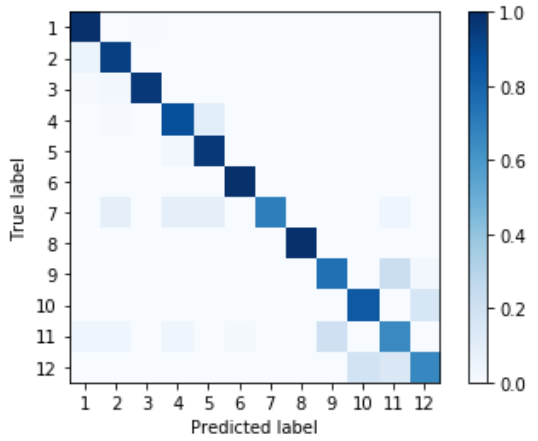
1) Préparation environnement & 1ère prédiction

```python
def logreg(trainx, trainy, testx, testy):
  from sklearn.linear_model import LogisticRegression
  time_start = time.clock()
  logres = LogisticRegression(solver='lbfgs', multi_class="ovr", max_iter=1000)
  logres.fit(trainx, trainy.values.ravel())
  predictions = logres.predict(testx)
  i = 0
  accuracy =0
  for res in predictions:
    if res == testy.loc[i, "y_target"]:
      accuracy +=1
    i+=1
  accuracy = (accuracy/i)
  time_elapsed = (time.clock() - time_start)
  return [logres, accuracy, time_elapsed]

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]
```

```python
logres = logreg(train_x, train_y, test_x, test_y)[0]
disp = plot_confusion_matrix(logres, test_x, test_y,
  cmap=plt.cm.Blues,
  normalize="true",include_values=False)
```



- Premier modèle de test non paramétré

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

1) Préparation environnement & 1ère prédiction

1) Export du model

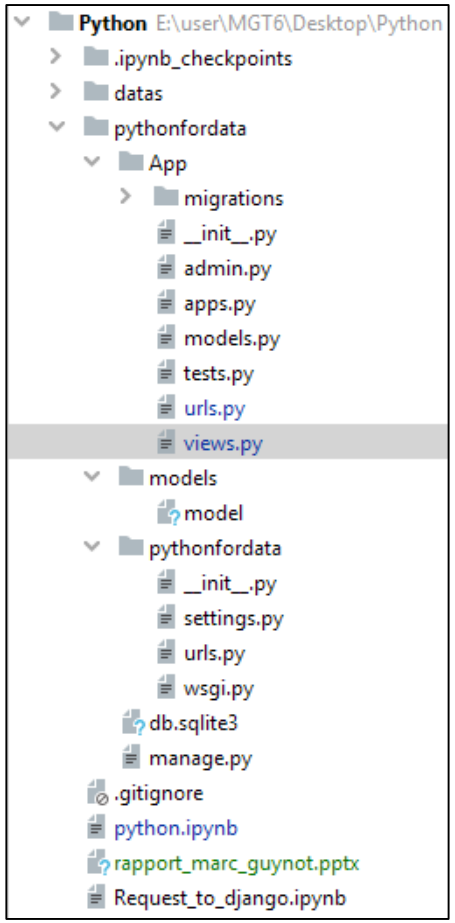
```
modellog = logreg(train_x, train_y, test_x, test_y)[0]
with open("pythonfordata/models/model", 'wb') as file:
            pickle.dump(modellog, file)
```

3) views.py : Class Predict

```
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)
```

2) Architecture Django



1) Préparation environnement & 1ère 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

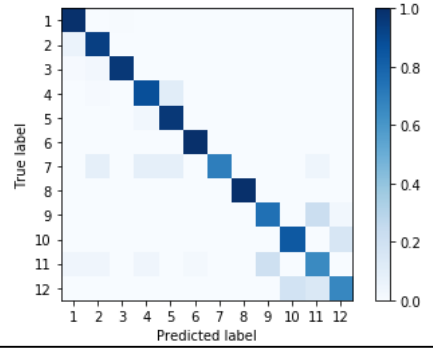
- Envoie de 5 observations
- Réponse en 18ms (serveur local), quelque soit le nombre d'observation envoyées

2) Models tunning

```python
def logreg(trainx, trainy, testx, testy):
  from sklearn.linear_model import LogisticRegression
  time_start = time.clock()
  logres = LogisticRegression(solver='lbfgs', multi_class="ovr",
                              max_iter=1000)
  logres.fit(trainx, trainy.values.ravel())
  predictions = logres.predict(testx)
  i = 0
  accuracy =0
  for res in predictions:
    if res == testy.loc[i, "y_target"]:
      accuracy +=1
    i+=1
  accuracy = (accuracy/i)
  time_elapsed = (time.clock() - time_start)
  confusion_matrix(testy, predictions)
  return [logres, accuracy, time_elapsed]

logres = logreg(train_x, train_y, test_x, test_y)
print("Précision : " + str(logres[1]) + ", temps : " + str(logres[2]) )
disp = plot_confusion_matrix(logres[0], test_x, test_y,
  cmap=plt.cm.Blues,
  normalize="true",include_values=False)
```

```
Précision : 0.9468690702087287, temps : 75.13321099999985
```

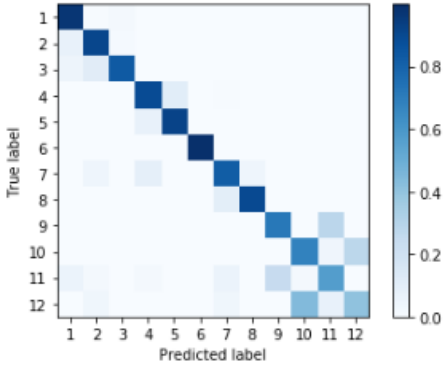


Regression logistique

```python
[13]
def rfc(trainx, trainy, testx, testy):
  from sklearn.ensemble import RandomForestClassifier
  time_start = time.clock()
  rfc = RandomForestClassifier(n_estimators = 200)
  rfc.fit(trainx,trainy.values.ravel())
  predictions = rfc.predict(testx)
  i = 0
  accuracy =0
  for res in predictions:
    if res == testy.loc[i, "y_target"]:
      accuracy +=1
    i+=1
  accuracy = (accuracy/i)
  time_elapsed = (time.clock() - time_start)
  return [rfc, accuracy, time_elapsed]

rf = rfc(train_x, train_y, test_x, test_y)
print("Précision : " + str(rf[1]) + ", temps : " + str(rf[2]) )
disp = plot_confusion_matrix(rf[0], test_x, test_y,
  cmap=plt.cm.Blues,
  normalize="true",include_values=False)
```

```
Précision : 0.9104996837444655, temps : 32.45343500000001
```

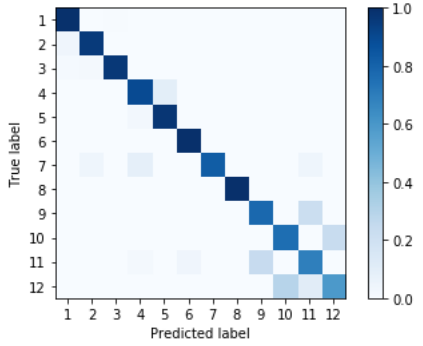


RandomForest Classifier

```python
def svm(trainx, trainy, testx, testy):
  from sklearn import svm
  time_start = time.clock()
  clf = svm.SVC(gamma=0.001, C=100., decision_function_shape='ovr',
                random_state=22)
  clf.fit(trainx, trainy.values.ravel())
  predictions = clf.predict(testx)
  i = 0
  accuracy =0
  for res in predictions:
    if res == testy.loc[i, "y_target"]:
      accuracy +=1
    i+=1
  accuracy = (accuracy/i)
  time_elapsed = (time.clock() - time_start)
  return [clf, accuracy, time_elapsed]

svm(train_x, train_y, test_x, test_y)[1]
sv = svm(train_x, train_y, test_x, test_y)
print("Précision : " + str(sv[1]) + ", temps : " + str(sv[2]) )
disp = plot_confusion_matrix(sv[0], test_x, test_y,
  cmap=plt.cm.Blues,
  normalize="true",include_values=False)
```

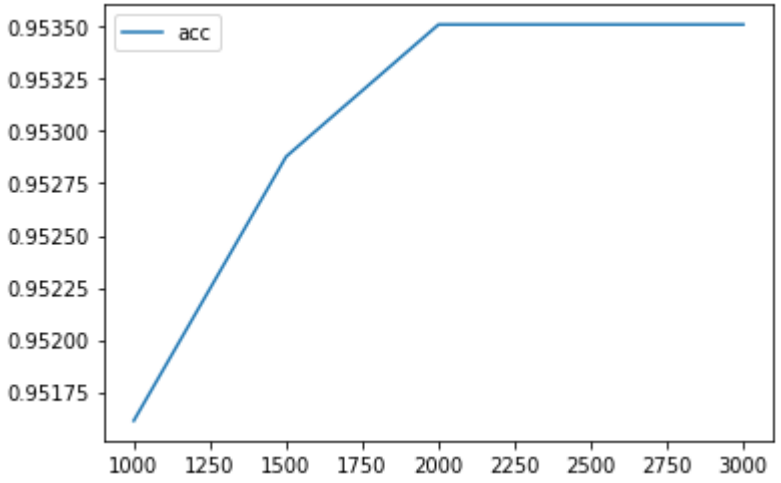
```
Précision : 0.9535104364326376, temps : 9.435485999999855
```



Support Vector Machines

2) Models tunning

Support Vector Machines Tunning



Max_iter

```
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]
```

Tolerence parameter

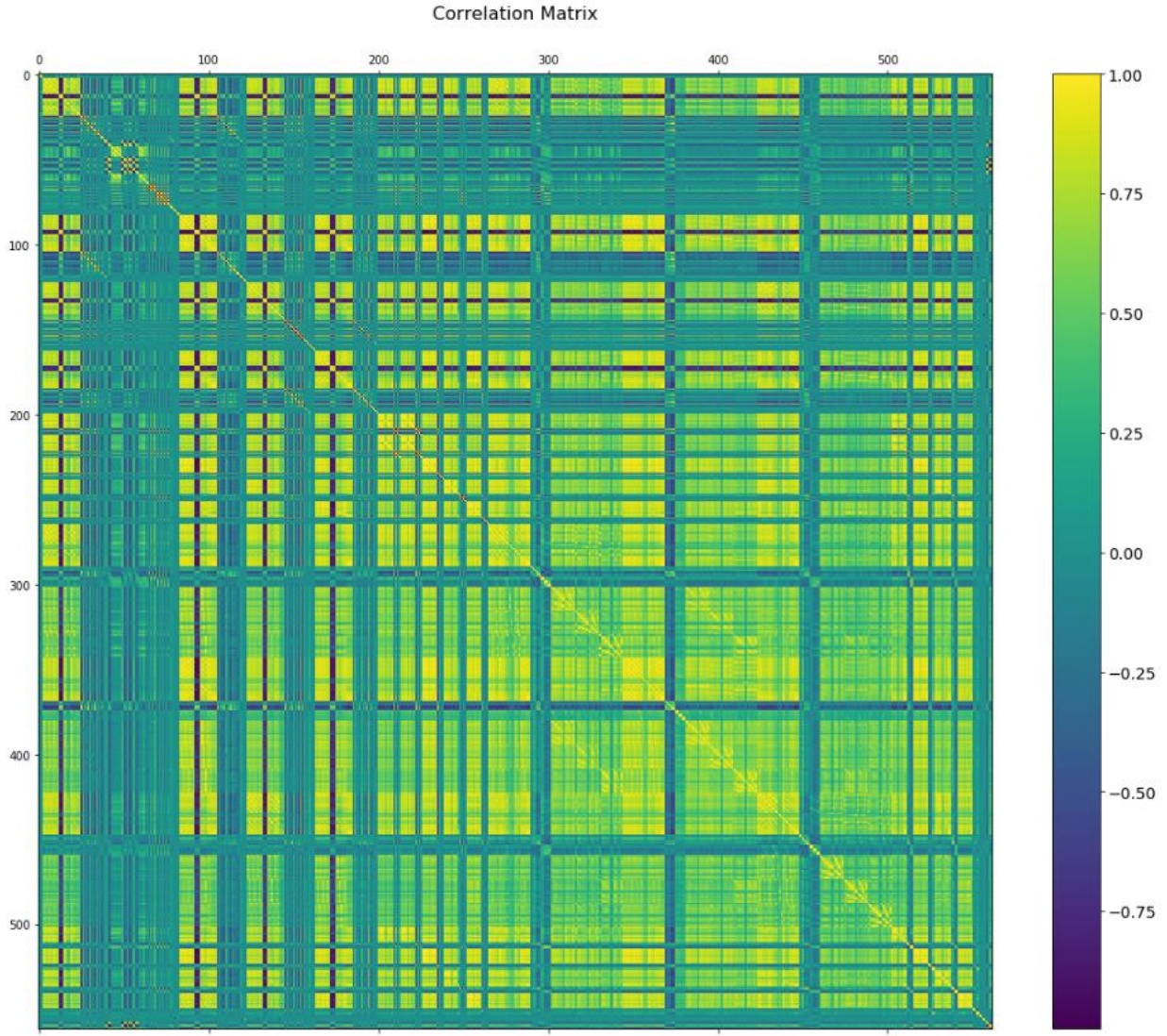
Paramètres retenus pour 95,38% :

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

Features engineering :

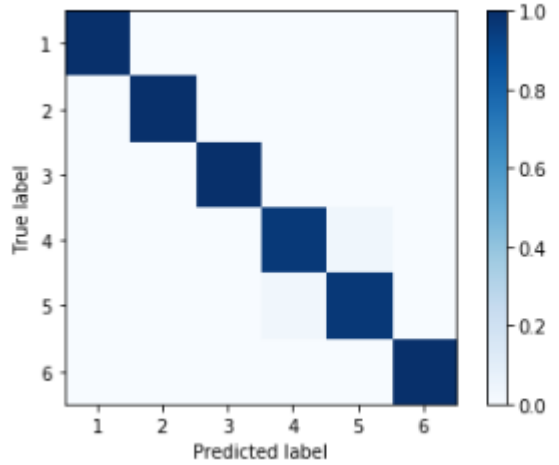
- En enlevant les postures intermédiaires qui sont présentes en sous nombre on améliore la prédiction.
- Ces features étaient indiqué comme « complémentaires » dans la présentation du dataset.

```
dfLight =  df[train['y_target'] < 7]
dfLight_x = trainLight.iloc[ : ,-1]
dfLight_y = trainLight.iloc[ : ,:-1]

trainLight_y, testLight_y, trainLight_x, testLight_x = train_test_split(
    dfLight_x, dfLight_y, test_size=0.30, random_state=42)
print("Nombre ligne initiale : " + str(df.shape[0]) +
      "\nNombre de ligne sans postures intermédiaires : " + str(dfLight.shape[0]))
```

```
Nombre ligne initiale : 10929
Nombre de ligne sans postures intermédiaires : 10414
```

```
Précision : 0.9865168539325843, temps : 3.832502999999633
```



Modèle SVC : 98.7 % de précision

Conclusion

- DataSet très qualitatif, aucune nécessité de retravailler la données pour obtenir des premiers modèles satisfaisants.
- Avec le DataSet complet et un modèle Support Vector Machine nous avons une prédiction de 95.4%
- En retirant les classes complémentaires du dataset de départ on monte à 98.7% de précision
- L'intégration Django quant à elle a un temps de réponse de 18ms (en local) en moyenne quelque soit la quantité de prédiction demandée