P00406: Machine learning
Coursework
16036767 *VAST Maxime*

Facial Expressions Recognition: A binary classification problem

1. Introduction

a. Tools

As I’m familiar with the programming language Python, I decided to use it for this assignment. During my research about doing machine learning in Python, I discovered an open data science platform called Anaconda (Continuum, 2017), which offers an environment with all the required scientific tools.
One of the tools available in Anaconda is SciKit-Learn (Scikit-learn.org, 2017) which is an open source Python machine learning kit using NumPy, SciPy, and matplotlib.

Although I will present some relevant parts of my code in this document, the entire program will be available as multiple separate Python files in this submission.
To make it work, one needs to install Anaconda (Python 3 version), and then using the embedded package manager ‘conda’, to install SciKit-Learn.
Then, the following command can be used:

```
$ /path/to/anaconda/bin/python main.py /path/to/data/folder
```

Will train the two chosen classifiers on “user a“ and test them on “user b”.

```
$ /path/to/anaconda/bin/python comparison.py /path/to/data/folder
```

Will compare all the classifiers on all the Grammatical Facial Expressions (GFE).

```
$ /path/to/anaconda/bin/python mlp-vs-svc.py.py /path/to/data/folder
```

Will compare Multi Layer Perceptron (MLP) against Support Vector Classifier (SVC) on the “topics” GFE.

b. Handling the data

The two following classes help in storing the data in Python objects. A point object being a triplet of coordinates, and a Frame object being composed of a timestamp, a class (expression is present or not) and a list of 99 points.

The classifier requires for two things in order to train:

- a two-dimensional array of features
- an uni-dimensional array of targets.

In order to remove our third dimension without losing information, we need to flatten our list points into a simple list. This will be done by the “flatten” function:

```python
#!/usr/bin/env python3

class Frame:
    def __init__(self, timestamp, binary_class):
        self.timestamp    = timestamp
        self.binary_class = binary_class
        self.points       = []

    def flatten(self):
        """ Convert a list of points to a list of numbers as classifiers
        don't handle 3d matrix
        [(0, 1, 2), (3, 4, 5)] becomes [0, 1, 2, 3, 4, 5]
        """
        lst = []
        for point in self.points:
            lst.extend([point.x, point.y, point.z])
        return lst
```

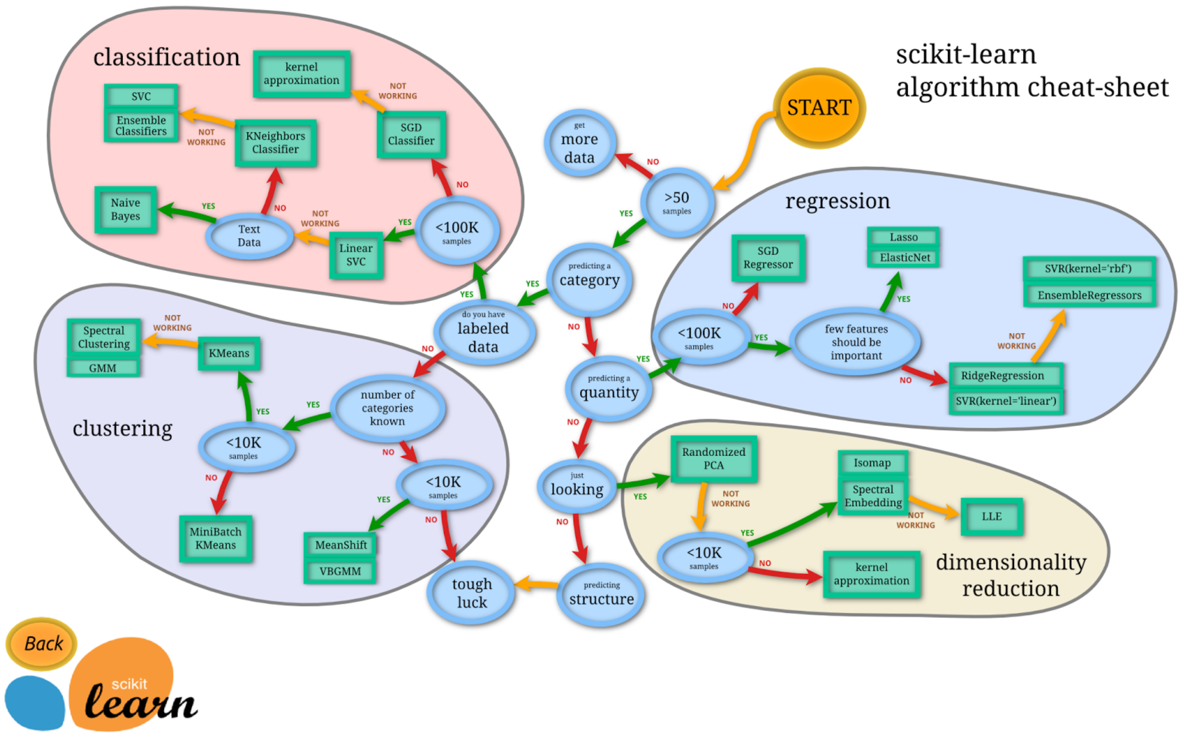
```python
class Point:
    def __init__(self, point_id, x, y, z):
        """ Initialize instance
        :param  x: Coordinate in pixels
        :type   x: double.
        :param  y: Coordinate in pixels
        :type   y: double.
        :param  z: Coordinate in millimeters
        :type   z: double.
        """
        self.id = point_id
        self.x  = x
        self.y  = y
        self.z  = z
```

The following function will use the two previously defined classes to transform a pair of 'datapoints' and the corresponding 'targets' files onto usable python objects.

```python
def build_data_set(path):
    datapoints_file = path + '_datapoints.txt'
    targets_file    = path + '_targets.txt'
    frames          = []
    # Read the target file to load per-frame binary class
    with open(targets_file) as f:
        binary_classes = f.readlines()
    binary_classes = [cl.strip() for cl in binary_classes]
    with open(datapoints_file) as f:
        # Bypass the headers
        for _ in range(1):
            next(f)
        for line_number, line in enumerate(f):
            tokens    = line.strip().split(' ')
            timestamp = tokens[0]
            tokens    = tokens[1:]  # Remove the timestamp from the coordinates list
            frame = Frame(timestamp, int(binary_classes[line_number]))
            point_id = 0
            counter  = 0
            coords   = {
                'x': None,
                'y': None,
                'z': None
            }
            for token in tokens:
                if counter == 0:
                    coords['x'] = token
                    counter    += 1
                elif counter == 1:
                    coords['y'] = token
                    counter    += 1
                elif counter == 2:
                    coords['z'] = token
                          point  = Point(point_id, float(coords['x']), float(coords['y']),
float(coords['z']))
                    frame.points.append(point)
                    counter     = 0
                    point_id   += 1
            frames.append(frame)
    return frames
```

2. Choosing the classifier and the sentence type

As we are doing classification, we will use the following tree to decide which algorithm must not be used:



(Scikit-learn.org, 2017)

In order to choose a classifier and decide which sentence we are going to classify, we will train and test the following classifiers on all the GFE:

- Multi Layer Perceptron (Artificial neural network)
- Logistic Regression CV (aka logit, MaxEnt) classifier
- K Nearest Neighbor (Pattern recognition)
- Support Vector Clustering (usually used in multi-class classification)
- Linear SVC (Support Vector Clustering using a linear kernel)
- Gaussian Naive Bayes
- Bernoulli Naive Bayes
- Stochastic Gradient Descent Classifier
- Gaussian Process Classifier

This approach has been chosen despite the fact that some of these classifiers are notoriously known to be inefficient with the kind of data we will use. For example, Naive Bayes are supposed to be adapted to text classification, and Gradient Descent to very large amounts of data. Although our dataset does not fulfil these conditions we can only be confident about an algorithm efficiency after having tested it.

a. Unscaled data

This first iteration will use the data as it has been given to us; without any uniformization. Results in Red are above 80%, results in Yellow are above 70.

i. Training and testing on “user a”

Using ‘SciKit-Learn’ ‘train_test_split’ (Scikit-learn.org, 2017) function, we can train 90 percent of the “user a” data and test it on the remaining 10%. The results are globally good, and LogisticRegCV reaches at least 80% for all the GFE types.

```
$ /opt/miniconda3/bin/python3.6 comparison.py data/grammatical_facial_expression
```

Sentence Type	MLP	LogisticRegCV	KNN	SVC	LinearSVC	GaussianNB	BernoulliNB	SGD	GaussianProcess
affirmative	0.6262	0.8037	0.7757	0.6355	0.4019	0.6916	0.6636	0.6168	0.7009
conditional	0.5602	0.9476	0.9215	0.6963	0.9581	0.9634	0.8168	0.6859	0.8168
doubt_question	0.8182	0.9242	0.9015	0.5985	0.9015	0.9091	0.8939	0.7879	0.6212
emphasis	0.7518	0.9716	0.8936	0.7872	0.8723	0.8652	0.7589	0.7021	0.7376
negative	0.531	0.9027	0.8673	0.6106	0.6991	0.7965	0.6372	0.6549	0.6195
relative	0.7511	0.9571	0.8798	0.6824	0.5794	0.9013	0.7811	0.7511	0.7768
topics	0.7389	0.9667	0.9167	0.7889	0.9111	0.8833	0.8278	0.8333	0.8222
wh_question	0.5039	0.876	0.7907	0.5814	0.8682	0.845	0.6589	0.7132	0.6822
yn_question	0.5971	0.8993	0.8705	0.6115	0.3885	0.8417	0.7986	0.8058	0.6978

```
[ 11:04am ] [ maxime@xps13:~/Devel/Brookes/MachineLearning/Coursework/ml/src ]
$
```

ii. Training and testing on “user b”

The process on “user b” gives very similar results:

```
$ /opt/miniconda3/bin/python3.6 comparison.py data/grammatical_facial_expression
```

Sentence Type	MLP	LogisticRegCV	KNN	SVC	LinearSVC	GaussianNB	BernoulliNB	SGD	GaussianProcess
affirmative	0.5185	0.8704	0.8519	0.5833	0.6852	0.7315	0.6944	0.5185	0.6019
conditional	0.6569	0.8922	0.8578	0.7304	0.8627	0.7549	0.7402	0.799	0.7059
doubt_question	0.7	0.8867	0.7667	0.54	0.8933	0.78	0.7133	0.7467	0.4533
emphasis	0.7259	0.9111	0.8	0.5852	0.8148	0.7407	0.763	0.4593	0.6074
negative	0.6352	0.8302	0.7547	0.5094	0.5723	0.7736	0.717	0.6855	0.5723
relative	0.6545	0.9634	0.822	0.6911	0.8743	0.801	0.6963	0.3613	0.7225
topics	0.8415	0.9399	0.8743	0.7705	0.7541	0.8743	0.7596	0.4317	0.7104
wh_question	0.7068	0.8421	0.8571	0.5564	0.782	0.8947	0.8496	0.8346	0.6466
yn_question	0.546	0.8391	0.7011	0.5632	0.8563	0.7759	0.6609	0.6954	0.5747

```
[ 11:10am ] [ maxime@xps13:~/Devel/Brookes/MachineLearning/Coursework/ml/src ]
```

iii. Training on “user a” and testing “user b”

However, when training all the classifiers on “user a” and testing them on “user b”, the results drastically drop. This phenomenon is well known on supervised learning and is called overfitting:

```
$ /opt/miniconda3/bin/python3.6 comparison.py data/grammatical_facial_expression
```

Sentence Type	MLP	LogisticRegCV	KNN	SVC	LinearSVC	GaussianNB	BernoulliNB	SGD	GaussianProcess
affirmative	0.635	0.5084	0.513	0.5084	0.5093	0.5084	0.5074	0.5084	0.5084
conditional	0.4385	0.5772	0.6947	0.7104	0.4735	0.4641	0.6691	0.3525	0.7104
doubt_question	0.5003	0.6466	0.4609	0.479	0.664	0.652	0.5418	0.5758	0.479
emphasis	0.6057	0.3929	0.5387	0.6049	0.4077	0.4003	0.4249	0.4107	0.6049
negative	0.4374	0.4905	0.6087	0.5499	0.5499	0.5499	0.6308	0.5152	0.5499
relative	0.3314	0.5084	0.6633	0.7111	0.3267	0.6418	0.6003	0.678	0.7111
topics	0.52	0.2773	0.6384	0.7441	0.5008	0.7858	0.2236	0.7436	0.7441
wh_question	0.4322	0.5414	0.5279	0.5866	0.573	0.5866	0.5354	0.5663	0.5866
yn_question	0.4551	0.4039	0.5898	0.5886	0.416	0.5846	0.6018	0.5886	0.5886

```
[ 11:20am ] [ maxime@xps13:~/Devel/Brookes/MachineLearning/Coursework/ml/src ]
```

b. Scaling the data

From the previous results, there is no classifier standing out from the crowd. The data representation is partially responsible for this situation. Even if the disposition of the points composing each frame are similar in the way they are grouped together, each column has it’s own and different data range. This leads the employed classifiers to attribute a weight to each coordinates and then consider some as being more important than others.

We are going to use ‘SciKit-Learn’ MinMaxScaler to force the data to use the same scale:

```
def scale(self):
        train_scaler = MinMaxScaler()
        test_scaler  = MinMaxScaler()
        self.train_features  = train_scaler.fit_transform(self.train_features)
        self.test_features   = test_scaler.fit_transform(self.test_features)
```

More explanations and a demonstration of the scaling efficiency can be found in part 5 (“Data scaling”).

i. Training and testing on “user a”

Again, the algorithms used on “user a” gives quite good results. We can observe that the Gaussian Naive Bayes Classifier and the LogisticRegCV results are not the best anymore. Other algorithms, however, seem to have taken advantage of the data normalization.

$ /opt/miniconda3/bin/python3.6 comparison.py data/grammatical_facial_expression

Sentence Type	MLP	LogisticRegCV	KNN	SVC	LinearSVC	GaussianNB	BernoulliNB	SGD	GaussianProcess
affirmative	0.757	0.7196	0.7944	0.785	0.5607	0.4953	0.7103	0.4766	0.5981
conditional	0.9372	0.9686	0.9686	0.9372	0.9634	0.6283	0.7592	0.9005	0.9476
doubt_question	0.8561	0.7803	0.9091	0.8485	0.9242	0.9015	0.8636	0.8106	0.947
emphasis	0.8794	0.4184	0.8723	0.8652	0.9078	0.4965	0.7872	0.8511	0.8936
negative	0.7788	0.6549	0.9115	0.7965	0.6726	0.5398	0.6991	0.5575	0.7345
relative	0.9227	0.3519	0.9313	0.9227	0.2704	0.7296	0.7468	0.9056	0.9313
topics	0.9333	0.8278	0.9278	0.8556	0.7556	0.8111	0.8278	0.9667	0.6444
wh_question	0.7597	0.5659	0.5581	0.876	0.5659	0.5891	0.6124	0.7519	0.8605
yn_question	0.8345	0.964	0.8705	0.8129	0.9281	0.9209	0.8201	0.9209	0.8993

[11:13am] [maxime@xps13:~/Devel/Brookes/MachineLearning/Coursework/ml/src]

ii. Training and testing on “user b”

The same observation can be done on the “user b”. The results seem even more satisfying, and the Multi Layer Perceptron goes beyond 80% for each of the GFE sentences.

$ /opt/miniconda3/bin/python3.6 comparison.py data/grammatical_facial_expression

Sentence Type	MLP	LogisticRegCV	KNN	SVC	LinearSVC	GaussianNB	BernoulliNB	SGD	GaussianProcess
affirmative	0.8241	0.6852	0.9352	0.7593	0.7407	0.463	0.7685	0.8426	0.8704
conditional	0.8186	0.6814	0.8284	0.8088	0.8186	0.75	0.75	0.8627	0.8922
doubt_question	0.8533	0.8933	0.9133	0.8667	0.92	0.5467	0.7267	0.82	0.9
emphasis	0.9259	0.8593	0.8963	0.8815	0.7556	0.6	0.763	0.837	0.9037
negative	0.8176	0.5723	0.9119	0.8616	0.7358	0.4088	0.7736	0.7044	0.8805
relative	0.822	0.8953	0.9005	0.8691	0.9215	0.6963	0.644	0.801	0.9215
topics	0.8962	0.7596	0.8743	0.8798	0.9563	0.6995	0.8033	0.8634	0.8852
wh_question	0.8346	0.7744	0.8722	0.8872	0.7293	0.4361	0.8496	0.8571	0.9098
yn_question	0.8218	0.4655	0.8736	0.8448	0.4713	0.546	0.7126	0.6207	0.8966

[11:16am] [maxime@xps13:~/Devel/Brookes/MachineLearning/Coursework/ml/src]

iii. Training on a and testing b

It is now time to test the classifiers on realistic conditions. We can still observe overfitting but the results of training each algorithm on “user a” and testing it on “user b” are better with uniformization than without.

$ /opt/miniconda3/bin/python3.6 comparison.py data/grammatical_facial_expression

Sentence Type	MLP	LogisticRegCV	KNN	SVC	LinearSVC	GaussianNB	BernoulliNB	SGD	GaussianProcess
affirmative	0.6089	0.5642	0.5233	0.5326	0.6583	0.5084	0.5102	0.6415	0.5093
conditional	0.7788	0.7124	0.7443	0.7458	0.7094	0.7148	0.7124	0.7886	0.7458
doubt_question	0.8283	0.8504	0.6025	0.7589	0.8403	0.682	0.5798	0.8297	0.674
emphasis	0.6711	0.5856	0.4829	0.6332	0.6257	0.4442	0.4539	0.5283	0.5231
negative	0.469	0.5664	0.4437	0.5082	0.5601	0.5499	0.5601	0.5057	0.4248
relative	0.8309	0.2946	0.7642	0.7668	0.291	0.7258	0.6943	0.8482	0.7925
topics	0.8838	0.3107	0.7671	0.7918	0.274	0.7436	0.3715	0.806	0.7682
wh_question	0.5994	0.5866	0.5828	0.7756	0.5866	0.5866	0.5444	0.6837	0.5723
yn_question	0.6858	0.6697	0.6415	0.6162	0.6415	0.5886	0.5961	0.6646	0.6323

[11:24am] [maxime@xps13:~/Devel/Brookes/MachineLearning/Coursework/ml/src]

The results are far from perfect, but the objective of this coursework is to test and train two classifiers on one kind of GFE.
Moreover, “a general-purpose universal optimization strategy is theoretically impossible, and the only way one strategy can outperform another is if it is specialized to the specific problem under consideration” (Ho and Pepyne, 2002)

From the previous results we can now decide which algorithm are going to be used, and on which GFE type. The 88% reached by The Multi Layer Perceptron with the “topics” sentence leads us to chose this classifier with this sentence. The second best result with “topics” is given by the Stochastic Gradient Descent Classifier.

iv. SGDClassifier is inconsistent

But as we can observe in the following screen capture, the SGDC algorithm does not give constant results for the chosen sentence. This is probably due to the random seed generated by scikit learn.

```
 $ /opt/miniconda3/bin/python3.6 mlp-vs-svc.py data/grammatical_facial_expression

Sentence Type           MLP             SGD
affirmative              0.6201          0.6415
conditional              0.7719          0.7935
doubt_question           0.8343          0.8223
emphasis                 0.6674          0.4457
negative                 0.4728          0.4608
relative                 0.7862          0.8503
topics                   0.8816          0.4603
wh_question              0.7297          0.637
yn_question              0.6789          0.668
[ 11:31am ]  [ maxime@xps13:~/Devel/Brookes/MachineLearning/Coursework/ml/src ]
 $ /opt/miniconda3/bin/python3.6 mlp-vs-svc.py data/grammatical_facial_expression

Sentence Type           MLP             SGD
affirmative              0.6201          0.5838
conditional              0.7719          0.763
doubt_question           0.8343          0.8123
emphasis                 0.6674          0.5201
negative                 0.4728          0.5019
relative                 0.7862          0.8293
topics                   0.8816          0.3211
wh_question              0.7297          0.6265
yn_question              0.6789          0.6922
[ 11:36am ]  [ maxime@xps13:~/Devel/Brookes/MachineLearning/Coursework/ml/src ]
```

We could have decided to change the default parameters of the algorithm, but as it seems to give constant results for “doubt question” and “relative” sentences, this algorithm may not be the best choice. The adopted solution was to select the third best classifier for “topics”, namely Support Vector Classifier.

v. SVC is consistent

Support vector Classifier returns the same results over multiple iterations. It is also a coherent choice with the tree presented in the beginning of part 2.

```
$ /opt/miniconda3/bin/python3.6 mlp-vs-svc.py data/grammatical_facial_expression

Sentence Type           MLP             SVC
affirmative              0.6201          0.5326
conditional              0.7719          0.7458
doubt_question           0.8343          0.7589
emphasis                 0.6674          0.6332
negative                 0.4728          0.5082
relative                 0.7862          0.7668
topics                   0.8816          0.7918
wh_question              0.7297          0.7756
yn_question              0.6789          0.6162
[ 11:48am ]  [ maxime@xps13:~/Devel/Brookes/MachineLearning/Coursework/ml/src ]
```

3. Main process

Every result of this section will be analyzed in part 4 !

a. Support Vector Classifier

i. Training and testing on "user a"

```
1 : start = time.time()
2 :
3 : algorithm_obj = SVC()
4 : classifier = SklearnClassifier()
5 : classifier.load_from_file('data/grammatical_facial_expression', 'topics')
6 : classifier.set_data(training='a', testing='a')
7 : classifier.scale()
8 : classifier.use(algorithm_obj)
10 : classifier.fit()
11 : score = classifier.score(classifier.test_features, classifier.test_classes)
12 : classes_pred = classifier.predict(classifier.test_features)
13 :
14 : print("SVC" + ' : ' + str(score))
15 : print('Elapsed time to train and test : ' + str(time.time() - start))
16 : print(classification_report(classifier.test_classes, classes_pred))
17 : print(confusion_matrix(classifier.test_classes, classes_pred))
```

Because line 6 sets the training and testing data to be both equal to "user a", the "set_data" function will call SciKit-Learn "train_test_split" (Scikit-learn.org, 2017) which will randomly split "user a" features as follows:

- 90% of the features is used as training data
- The remaining 10% of the features are used as testing data

```python
def set_data(self, training='a', testing='b'):
    X_train  = np.array([f.flatten() for f in self.data[training]])
    y_train  = np.array([f.binary_class for f in self.data[training]])
    X_test   = np.array([f.flatten() for f in self.data[testing]])
    y_test   = np.array([f.binary_class for f in self.data[testing]])

    if training == testing:
        self.train_features,
        self.test_features,
        self.train_classes,
        self.test_classes = train_test_split(X_train, y_train, test_size=0.1)
    else:
        self.train_features = X_train
        self.train_classes  = y_train
        self.test_features  = X_test
        self.test_classes   = y_test
```

The result of the prints from line 14 to line 17 is the following:

```
SVC : 0.838888888889
Elapsed time to train and test : 1.84267258644104
             precision    recall  f1-score   support

          0       0.83      1.00      0.91       141
          1       1.00      0.26      0.41        39

avg / total       0.87      0.84      0.80       180

[[141   0]
 [ 29  10]]
[  1:21pm ]  [ maxime@xps13:~/Devel/Brookes/MachineLearning/Coursework/ml/src ]
```

ii. Training on “user a” and testing on “user b”

We just replaced line 6:

`classifier.set_data(training='a', testing='a')` by: `classifier.set_data()`

As you can see in the block code above, the “set_data” function will, by default, set “user a” features as the training data and “user b” features as the testing data.

The result of the prints from line 14 to line 17 is the following:

```
SVC : 0.791780821918
Elapsed time to train and test : 2.6276450157165527
             precision    recall  f1-score   support

          0       0.78      0.99      0.88      1358
          1       0.91      0.21      0.34       467

avg / total       0.82      0.79      0.74      1825

[[1348   10]
 [ 370   97]]
[  1:22pm ]  [ maxime@xps13:~/Devel/Brookes/MachineLearning/Coursework/ml/src ]
```

b. Multi Layer Perceptron

i. Training and testing on “user a”

We replaced line 3:

```
algorithm_obj = SVC()
```

by:

```
Algorithm_obj = MLPClassifier(solver='sgd', alpha=0.1, hidden_layer_sizes=(300,),
random_state=0, activation='relu', max_iter=2000, learning_rate='adaptive')
```

The random_state parameter has been set to 0 to avoid the changing behaviour previously observed with the Gradient descent. The other parameters have been progressively refined to reach to best possible result with this neural network.

The result of the prints from line 14 to line 17 is the following:

```
[  1:21pm ]  [ maxime@xps13:~/Devel/Brookes/MachineLearning/Coursework/ml/src ]
 $ /opt/miniconda3/bin/python3.6 main.py data/grammatical_facial_expression

MLP : 0.844444444444
Elapsed time to train and test : 4.326370477676392
             precision    recall  f1-score   support

          0       0.90      0.90      0.90       141
          1       0.64      0.64      0.64        39

avg / total       0.84      0.84      0.84       180

[[127  14]
 [ 14  25]]
```

ii. Training on “user a” and testing on “user b”

Again we replaced line 6 to set “user b” as the testing data.

The result of the prints from line 14 to line 17 is the following:

```
[  1:21pm ]  [ maxime@xps13:~/Devel/Brookes/MachineLearning/Coursework/ml/src ]
 $ /opt/miniconda3/bin/python3.6 main.py data/grammatical_facial_expression

MLP : 0.881643835616
Elapsed time to train and test : 5.206083059310913
             precision    recall  f1-score   support

          0       0.90      0.95      0.92      1358
          1       0.82      0.69      0.75       467

avg / total       0.88      0.88      0.88      1825

[[1288   70]
 [ 146  321]]
```

4. Results comparison and performances

a. Global performances

As a reminder, the following screen captures show the performance comparison between the two chosen algorithms on each sentence:

i. Training and testing on “user a”

```
$ /opt/miniconda3/bin/python3.6 mlp-vs-svc.py data/grammatical_facial_expression

entence Type            MLP              SVC
ffirmative               0.6636           0.8037
onditional               0.9424           0.9476
oubt_question            0.9167           0.9015
mphasis                  0.8652           0.8652
egative                  0.8053           0.5664
elative                  0.9142           0.9399
opics                    0.8611           0.9111
h_question               0.7752           0.814
n_question               0.9065           0.8633
  1:07pm ]  [ maxime@xps13:~/Devel/Brookes/MachineLearning/Coursework/ml/src ]
```

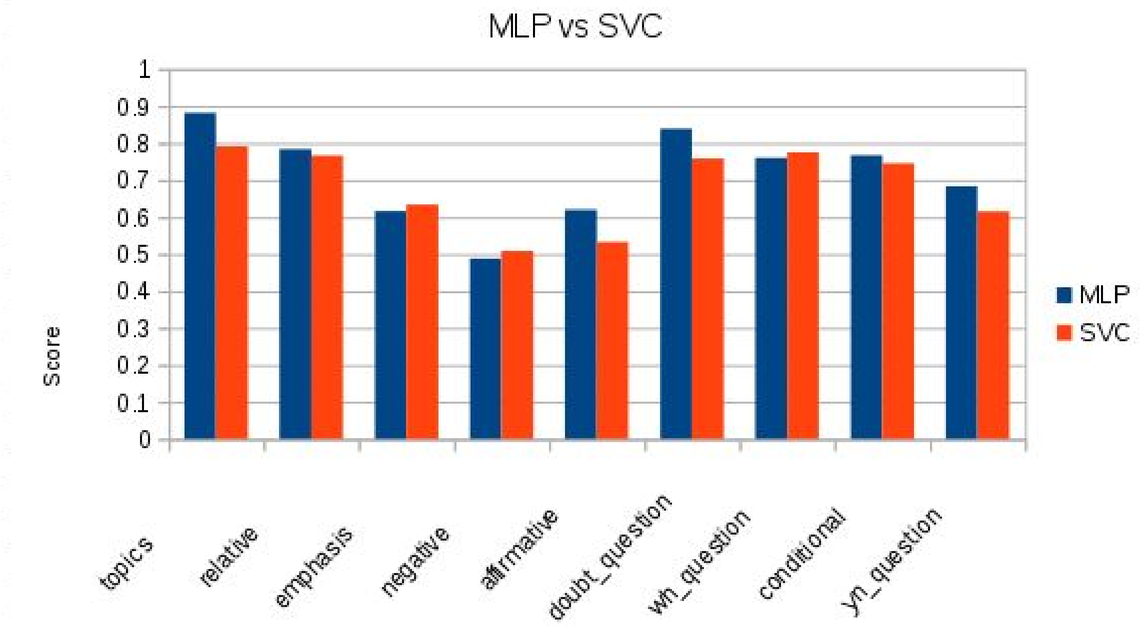
ii. Training and testing on “user b”

```
 $ /opt/miniconda3/bin/python3.6 mlp-vs-svc.py data/grammatical_facial_expression

Sentence Type            MLP              SVC
affirmative               0.8426           0.8611
conditional               0.8333           0.8235
doubt_question            0.8467           0.9067
emphasis                  0.9185           0.9259
negative                  0.761            0.8302
relative                  0.9058           0.9005
topics                    0.8525           0.8798
wh_question               0.8797           0.9023
yn_question               0.7011           0.8678
[  1:11pm ]  [ maxime@xps13:~/Devel/Brookes/MachineLearning/Coursework/ml/src ]
```

iii. Training on “user a” and testing on “user b”

```
 $ /opt/miniconda3/bin/python3.6 mlp-vs-svc.py data/grammatical_facial_expression

Sentence Type            MLP              SVC
affirmative               0.6201           0.5326
conditional               0.7675           0.7458
doubt_question            0.8283           0.7589
emphasis                  0.6168           0.6332
negative                  0.4874           0.5082
relative                  0.7841           0.7668
topics                    0.8816           0.7918
wh_question               0.7605           0.7756
yn_question               0.683            0.6162
[  1:15pm ]  [ maxime@xps13:~/Devel/Brookes/MachineLearning/Coursework/ml/src ]
```

This graph is showing the efficiency comparison of both algorithms with scaled data:



We can observe that even if MLP is better than SVC - and sometimes outperform it by almost 10% - it is not always the case. Indeed, SVC is better with “emphasis”, “negative” and “wh_question” sentences.

b. Results and performances on “topics” sentence

i. Testing and training on “User A”

1. Support Vector Classifier

Score : 0.838888888889
Elapsed time to train and test : 1.84267258644104 seconds

Performances:

	precision	recall	f1-score	support
Class 0	0.83	1.00	0.91	141
Class 1	1.00	0.26	0.31	39
avg / total	0.87	0.84	0.80	180

Confusion matrix:

	Predicted class 0	Predicted class 1
True class 0	141	0
True class 1	29	10

2. Multi Layer Perceptron

Score : 0.844444444444
Elapsed time to train and test : 4.326370477676392

Performances:

	precision	recall	f1-score	support
Class 0	0.90	0.90	0.90	141
Class 1	0.64	0.64	0.64	39
avg / total	0.84	0.84	0.84	180

Confusion matrix:

	Predicted class 0	Predicted class 1
True class 0	127	14
True class 1	14	25

ii. Training on “user A” and testing on “user B”

1. Support Vector Classifier

Score : 0.791780821918
Elapsed time to train and test : 2.6276450157165527

Performances:

	precision	recall	f1-score	support
Class 0	0.78	0.99	0.88	1358
Class 1	0.91	0.21	0.34	467
avg / total	0.82	0.79	0.74	1825

Confusion matrix:

	Predicted class 0	Predicted class 1
True class 0	1348	10
True class 1	370	97

2. Multi Layer Perceptron

Score : 0.881643835616
Elapsed time to train and test : 5.206083059310913

Performances:

	precision	recall	f1-score	support
Class 0	0.90	0.95	0.92	1358
Class 1	0.82	0.69	0.75	467
avg / total	0.88	0.88	0.88	1825

Confusion matrix:

	Predicted class 0	Predicted class 1
True class 0	1288	70
True class 1	146	321

The MLP is almost 10% better than the SVC Classifier but takes an average time to train and test that is twice as long as the SVC. Training time could be reduced “by partitioning the training task in multiple training subtasks with sub-models, which can be performed independently and in parallel”. (Miranda and Von Zuben, 2016)

As we can see on the two “**performance**” tables above, SVC is not far behind MLP on the class 0 (0.88 vs 0.92 on F1-score), but drops down on the class 1 (0.34 vs 0.75).

Comparing the confusion matrixs helps understanding what’s happening:

Classifier	True Positive	True Negative	False Negative	False Positive
SVC	1348	97	10	370
MLP	1288	321	70	146
Difference	-4.66 %	69.78 %	85.71 %	-153.42 %

“Difference” is calculated using a cross product to process the difference between SVC and MLP expressed in percentage.

$$Difference = 100 - \frac{(SVC*100)}{MLP}$$

As we can observe, SVC's False Positive score is responsible for the global score droping down.

iii. Metrics explanation

Classifier scores, effectiveness and accuracy are defined by some metrics : Precision, Recall and F1-Score.

1. Precision

Precision shows how many retrieved instances are relevant. Precision is defined by the following formula:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

With :

- True Positives (tp) : The representation of all the values that are correctly predicted. (condition and prediction are both positive)
- False Positives (fp) : The value is badly classified. (condition is negative and prediction is positive). Also known as “Type I error” : the prediction was good but has been rejected.

2. Recall

Recall (or sensitivity) represents the fraction of relevant instances that are retrieved.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

With:

- False Negatives (fn) : The value is badly classified. (condition is positive and prediction is negative). Also known as “Type II error” : the prediction was incorrect but has been accepted anyway.

3. F1-Score

F1-score is a measure of accuracy. It considers both the precision and the recall The general formula is called F-measure and is expressed by :

$$F_\beta = \frac{(1+\beta^2) \bullet (precision \bullet recall)}{(\beta^2 \bullet precision + recall)}$$

In our case, beta is set to 1 in order to prevent weight:

$$F_\beta = \frac{(precision \bullet recall)}{(precision + recall)}$$

4. Confusion Matrix

In the case of a binomial classification, the confusion matrix is a table with two rows and two columns that reports the number of false positives, false negatives, true positives, and true negatives. It has the following representation:

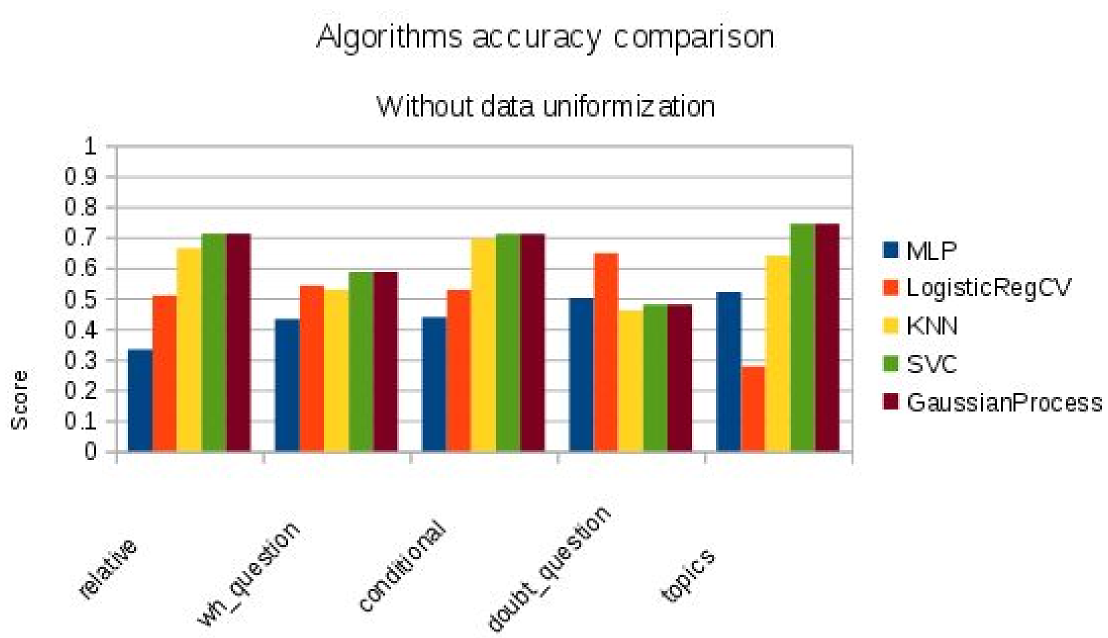
	Predicted class 0	Predicted class 1
True class 0	True Positive	False Negative
True class 1	False Positive	True Negative

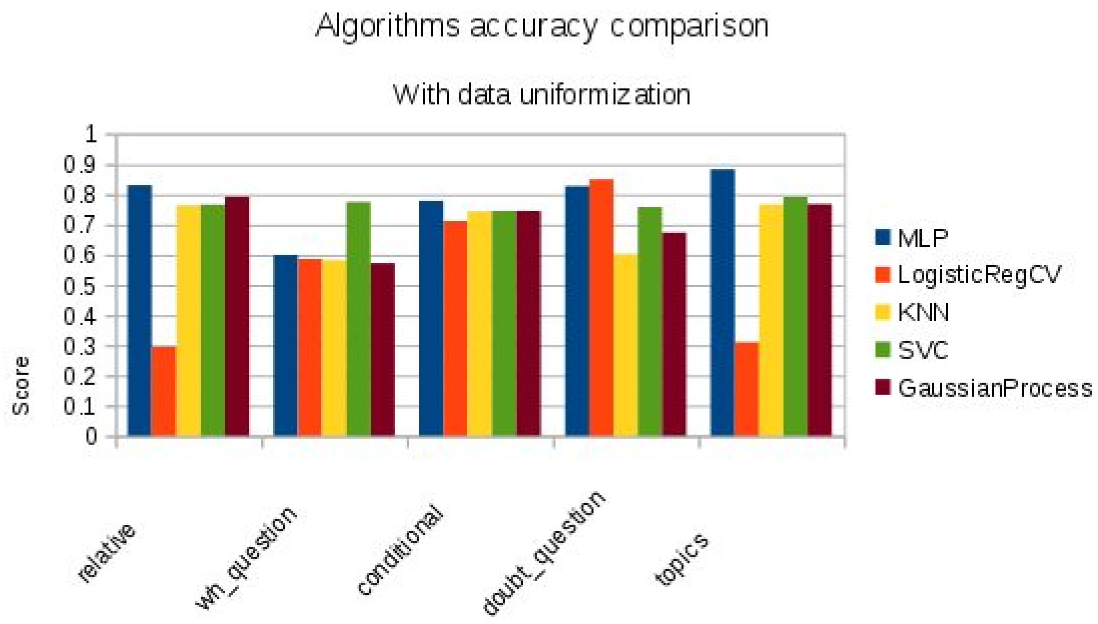
5. Support

The support is simply the number of occurrences of each class.

5. Data scaling

Data scale leverages Machine Learning results. As stated in part 2.b. ("Scaling the data"), the 3D representation of the GFE, and the way the data has been captured (from 2 different videos showing 2 different users), implies multiple data range that cannot be properly compared. Data uniformization and standardization is a preprocessing step that prepares the data before ingestion. It aims to bring all the variables on the same scale for the Machine Learning Algorithms to give the same importance to every data.
You will find below a comparison of 5 of the 9 Machine Learning Algorithms used in this assignment, running over 5 different data sets. Without any preprocessing, the average accuracy score is **0.57%** while it reaches **0.70%** after the min max scaler has been applied.





6. Conclusion

There is no such thing as a free lunch, Machine Learning is all about finding the right setup for the right data. Ultimately, this painful process we went through during this assignment must be automated. We observed how a tiny modification of a classifier parameters can have a big impact on the learning process. How the way the data is represented (scale) can make it more adapted to an algorithm rather than another. How some classifier with a random seed parameter can perform almost perfectly on one iteration and the next one become as useful as a coin toss. How the efficiency is constantly balanced with the processing time and the hardware resource. If I had to summarize my Machine learning experience so far; I would say that it all comes down to a process of “Trial and error”, to find the best trade-off between overfitting and inaccurate results.

7. References

- Continuum. (2017). Continuum. [online] Available at: https://www.continuum.io/ [Accessed 23 Apr. 2017].
- Scikit-learn.org. (2017). scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation. [online] Available at: http://scikit-learn.org [Accessed 23 Apr. 2017].
- Scikit-learn.org. (2017). 3.1. Cross-validation: evaluating estimator performance — scikit-learn 0.18.1 documentation. [online] Available at: http://scikit-learn.org/stable/modules/cross_validation.html [Accessed 23 Apr. 2017].
- Scikit-learn.org. (2017). Scikit-learn.org Machine learning map [online] Available at: http://scikit-learn.org/stable/_static/ml_map.png [Accessed 23 Apr. 2017].
- Ho, Y. and Pepyne, D. (2002). Simple Explanation of the No-Free-Lunch Theorem and Its Implications. Journal of Optimization Theory and Applications, 115(3), pp.549-570.
- Conrado S. Miranda, Fernando J. Von Zuben (2016). Reducing the Training Time of Neural Networks by Partitioning. arxiv.org arXiv:1511.02954
- En.wikipedia.org. (2017). Precision and recall. [online] Available at: https://en.wikipedia.org/wiki/Precision_and_recall [Accessed 23 Apr. 2017]