ILLINOIS INSTITUTE OF TECHNOLOGY

CS-584 Machine Learning Final Report

Cohen Raphaël
A20432518
December 8, 2018

INTRODUCTION

This project aims at making us better understand different classification methods as well as their parameters by fine tuning them to get the best performances possible. The dataset used as testbed for our experimentation consists of face images distributed among three major classes. Namely, "Bush", "Serena Williams" and "Other". As we are only interested in binary classification and this is why we will consider two classification problems, classifying Bush versus the other ones (including Serena Williams) and classifying Serena Williams versus the other ones (including Bush). One of the greatest challenges we will try to overcome for this project is the fact that Bush and Williams classes are highly unbalanced. In order to have meaningful results for the classification task we want to accomplish, we will use the *F1-score* metric throughout the entirety of this project. For the first two phases of this project, we will consider the results from two classifiers that are *KNN* and *SVC*, with and without prior *PCA* transformation. The last two phases focus on convolutional neural network models (*CNN*), with and without pre-training them first on a larger dataset.

PHASE 1

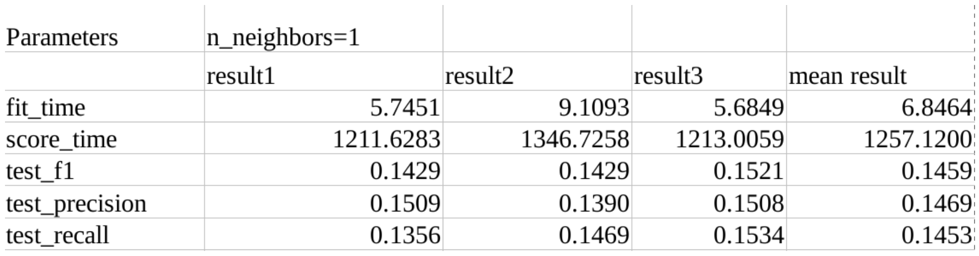
During this first phase, instructions were to focus on two specific classifiers. K-nearest neighbors (*KNN*) and Support vector machine (*SVM*). Both of them have their own set of parameters that we are going to explore to improve our classification performances. For *Phase 1* and *Phase 2* all our results will be obtained by using a *3-Fold Cross Validation*, with the *StratifiedKFold* parameter to respect the distribution of each class in our samples.

1.1 KNN

”The k-nearest neighbors algorithm (KNN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space.”
- *Wikipedia*

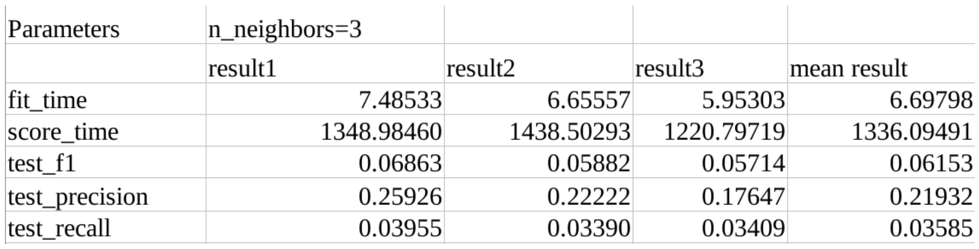
The *KNN algorithm* only has one parameter that we can modify and it is K, the number of nearest neighbors that we want to take into account when classifying a new object. For this project we will take $K \in \{1, 3, 5\}$. Since we have an even number of classes, we need an odd value for K. Moreover, as we only have 2 classes, we expect $K = 1$ to have the best performances.

RESULTS FOR BUSH



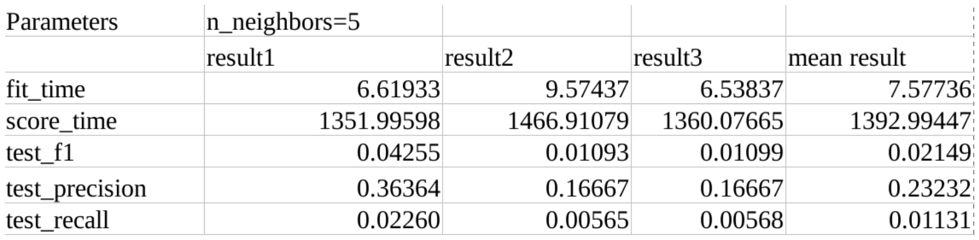
Parameters	n_neighbors=1			
	result1	result2	result3	mean result
fit_time	5.7451	9.1093	5.6849	6.8464
score_time	1211.6283	1346.7258	1213.0059	1257.1200
test_f1	0.1429	0.1429	0.1521	0.1459
test_precision	0.1509	0.1390	0.1508	0.1469
test_recall	0.1356	0.1469	0.1534	0.1453

Figure 1.1: KNN results for Bush with K = 1



Parameters	n_neighbors=3			
	result1	result2	result3	mean result
fit_time	7.48533	6.65557	5.95303	6.69798
score_time	1348.98460	1438.50293	1220.79719	1336.09491
test_f1	0.06863	0.05882	0.05714	0.06153
test_precision	0.25926	0.22222	0.17647	0.21932
test_recall	0.03955	0.03390	0.03409	0.03585

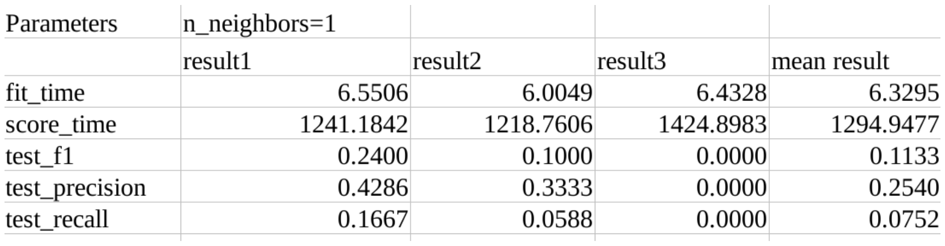
Figure 1.2: KNN results for Bush with K = 3



Parameters	n_neighbors=5			
	result1	result2	result3	mean result
fit_time	6.61933	9.57437	6.53837	7.57736
score_time	1351.99598	1466.91079	1360.07665	1392.99447
test_f1	0.04255	0.01093	0.01099	0.02149
test_precision	0.36364	0.16667	0.16667	0.23232
test_recall	0.02260	0.00565	0.00568	0.01131

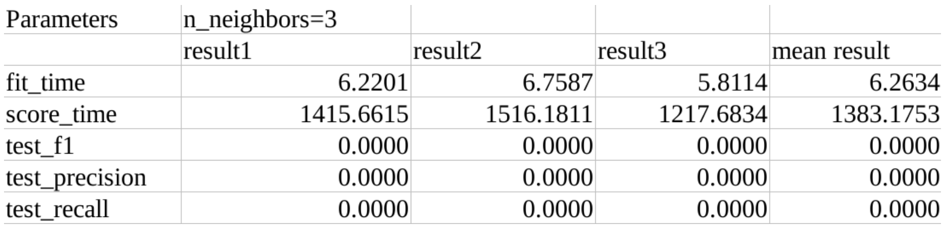
Figure 1.3: KNN results for Bush with K = 5

As we expected, we can see that $K = 1$ gave us the highest *F1-score* with a mean $F1 = 0.1459$



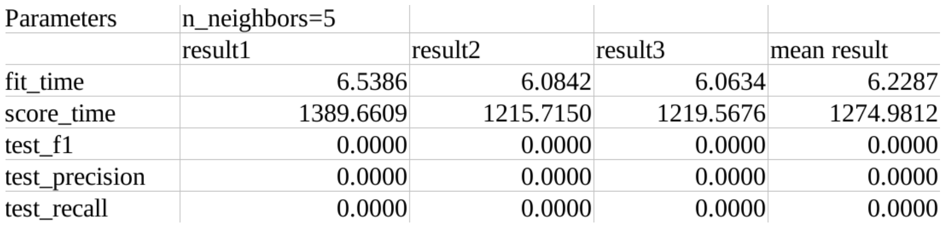
Parameters	n_neighbors=1			
	result1	result2	result3	mean result
fit_time	6.5506	6.0049	6.4328	6.3295
score_time	1241.1842	1218.7606	1424.8983	1294.9477
test_f1	0.2400	0.1000	0.0000	0.1133
test_precision	0.4286	0.3333	0.0000	0.2540
test_recall	0.1667	0.0588	0.0000	0.0752

Figure 1.4: KNN results for Williams with K = 1



Parameters	n_neighbors=3			
	result1	result2	result3	mean result
fit_time	6.2201	6.7587	5.8114	6.2634
score_time	1415.6615	1516.1811	1217.6834	1383.1753
test_f1	0.0000	0.0000	0.0000	0.0000
test_precision	0.0000	0.0000	0.0000	0.0000
test_recall	0.0000	0.0000	0.0000	0.0000

Figure 1.5: KNN results for Williams with K = 3



Parameters	n_neighbors=5			
	result1	result2	result3	mean result
fit_time	6.5386	6.0842	6.0634	6.2287
score_time	1389.6609	1215.7150	1219.5676	1274.9812
test_f1	0.0000	0.0000	0.0000	0.0000
test_precision	0.0000	0.0000	0.0000	0.0000
test_recall	0.0000	0.0000	0.0000	0.0000

Figure 1.6: KNN results for Williams with K = 5

Same conclusion for Williams here, $K = 1$ gave us the highest *F1-score* with a mean $F1 = 0.1133$

1.2 SVM

”An *SVM* model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.”
- *Wikipedia*

As *SVMs* have multiple parameters, we won’t be able to exhaustively test every one of them here. Those to be considered for this phase are described below.

- **Kernels:** How to compare two observations when the decision boundary is not always linear.
  - Linear

  - Polynomial (with a degree parameter)
  - Sigmoid
  - Radial Basis Function

- **Gamma parameter:** defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'.
- **C parameter:** trades off correct classification of training examples against maximization of the decision function's margin.

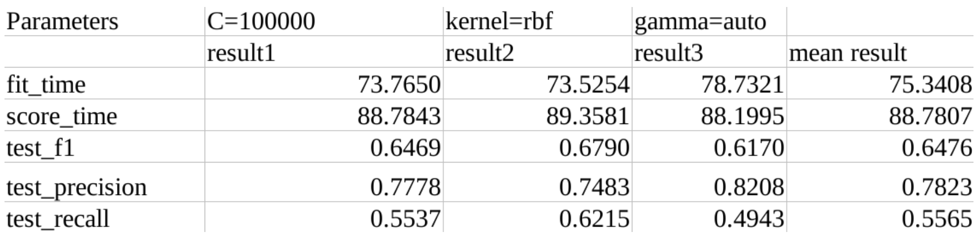
As a lot of our $F1$ values were zeros, I will only showcase a summary of the parameters that resulted in a zero value for $F1$, as well as the single best combination of parameters in terms of $F1$.

RESULTS FOR BUSH

Parameters that resulted in a zero F1-score

Kernel	Gamma	C	Degree
Linear	N/A	0.001	N/A
Polynomial	auto/scale	0.001, 1, 1000	1,2,3
Sigmoid	auto/scale	0.001, 1	N/A
RBF	auto/scale	0.001, 1, 1000	N/A

(1)



Parameters	C=100000	kernel=rbf	gamma=auto	
	result1	result2	result3	mean result
fit_time	73.7650	73.5254	78.7321	75.3408
score_time	88.7843	89.3581	88.1995	88.7807
test_f1	0.6469	0.6790	0.6170	0.6476
test_precision	0.7778	0.7483	0.8208	0.7823
test_recall	0.5537	0.6215	0.4943	0.5565

Figure 1.7: Best parameters in terms of F1 for Bush

The final mean *F1-score* with the *SVM* classifier is a massive improvement from what we had with *KNN*. Of course, this is only if the correct parameters are used for the *SVM*.

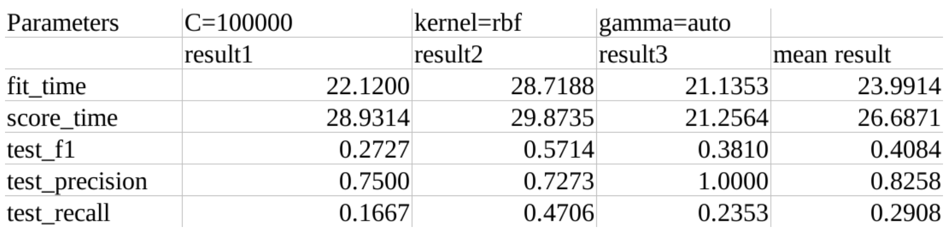
	Prev Best F1 (KNN-1)	New Best F1 (SVM)
Mean F1	0.14594	0.6476

(2)

Parameters that resulted in a zero F1-score

Kernel	Gamma	C	Degree
Linear	N/A	0.001	N/A
Polynomial	auto/scale	0.001, 1, 1000	1,2,3
Sigmoid	auto/scale	0.001, 1, 1000	N/A
RBF	auto/scale	0.001, 1, 1000	N/A

(3)



Parameters	C=100000	kernel=rbf	gamma=auto	
	result1	result2	result3	mean result
fit_time	22.1200	28.7188	21.1353	23.9914
score_time	28.9314	29.8735	21.2564	26.6871
test_f1	0.2727	0.5714	0.3810	0.4084
test_precision	0.7500	0.7273	1.0000	0.8258
test_recall	0.1667	0.4706	0.2353	0.2908

Figure 1.8: Best parameters in terms of F1 for Williams

Same comment for *Williams*, great improvement over *KNN* with the right parameters.

	Prev Best F1 (KNN-1)	New Best F1 (SVM)
Mean F1	0.1133	0.4084

(4)

PHASE 2

The reason for this phase is to test whether or not we should use *PCA* transformation on our data for either, improving our *F1-score* or simply reduce the computation time by reducing our problem's dimensions.

> ”Principal component analysis (*PCA*) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components.”
> - *Wikipedia*

Why should we use *PCA* ? The goal is to find a new base in which to project our features. A base that maximizes variance explained on the first few base vectors. By doing so, we have the

majority of our data variance expressed with only a few number of principal components. This technique has two major strengths. First it reduces the size of the problem so computations are faster. Second, it reduces the noise inside the data which in turn reduces the variance of our model and could even improve its performances.
How to chose *PCA* dimensions ? I calculated the cumulative sum of the explained variance for each of those number of dimensions: $\{64, 128, 256, 512, 1024\}$ and selected those who had a score above 90%. The last two, 512 and 1024, were selected to be tested during this phase.

For readability considerations, I will only be displaying the best scores and best parameters from both, *KNN* and *SVM* classifiers

2.1 KNN

Results for Bush

These results were obtained with K = 1.

	No PCA	PCA dim = 512
Mean F1	0.14594	0.14659

(5)

Even thought it is probably not significant, we can observe a very slight F1 improvement when using *PCA* transformation with *KNN*. However, the computation part went a lot faster.

Results for Williams

These results were obtained with K = 1.

	No PCA	PCA dim = 512
Mean F1	0.11333	0.13431

(6)

Same interpretation as previously. Very slight performance improvement.

2.2 SVM

In this part, the results were not as I expected. First of all, the best parameters no longer were: $C = 100000, Kernel = rbf, Gamma = auto$, but instead: $C = 1.0, Kernel = linear, Gamma = auto$ for both, *Bush* and *Williams*. Secondly, the *F1-score* actually decreased when using the *PCA* transformation for the *Bush* case.

Results for Bush

	No PCA	PCA dim = 1024
Mean F1	0.64763	0.0.51976

(7)

We can see that, this time the loss is significant as opposed to the KNN model.

Results for Williams

	No PCA	PCA dim = 1024
Mean F1	0.408369	0.50536

(8)

Pleasantly enough, the results for *Williams* were both, better and faster to compute, with *PCA* than without.

Phase 3

For this phase, the objective was to build a deep neural network model with keras, that would try to classify *Bush* and *Williams* images. Convolutional Neural Networks (*CNN*) are often used when it comes to image recognition and this is why I decided to use this type of architecture for my model.

The model

Our dataset is relatively small and have huge class imbalance. This is why the model needs to be small enough to avoid overfitting. As for the activation functions I tried both, Relu and Tanh and Tanh consistently gave me better results so this is the one I went for. We have two classes and we want a single output that we can interpret as a probability which means that the sigmoid function is the best fit for the output layer.
Here is a diagram of the model:

```
    OPERATION           DATA DIMENSIONS   WEIGHTS(N)   WEIGHTS(%)

        Input   #####     64   64    1
       Conv2D    \|/  -------------------       160     1.3%
         tanh   #####     64   64   16
 MaxPooling2D   Y max -------------------         0     0.0%
                #####     32   32   16
       Conv2D    \|/  -------------------      1160     9.5%
         tanh   #####     32   32    8
 MaxPooling2D   Y max -------------------         0     0.0%
                #####     16   16    8
      Dropout    | || -------------------         0     0.0%
                #####     16   16    8
       Conv2D    \|/  -------------------       584     4.8%
         tanh   #####     16   16    8
 MaxPooling2D   Y max -------------------         0     0.0%
                #####      8    8    8
      Flatten   ||||| -------------------         0     0.0%
                #####        512
      Dropout    | || -------------------         0     0.0%
                #####        512
        Dense   XXXXX -------------------     10260    84.2%
         tanh   #####         20
        Dense   XXXXX -------------------        21     0.2%
      sigmoid   #####          1
```

Figure 3.9: Phase 3: model architecture

The training

In order to train this model correctly while avoiding as much as possible over-fitting, I added some dropout layers to force the network to generalize its learning. At first, I wasn't getting good results because of the class imbalance. However, once I started using weights for my classes, it immediately worked better. I used the negative class as reference for the weights, which means that it had a weight of 1 while the positive class (Bush or Williams) had a weight of $\frac{Numberofnegatives}{Numberofpositives}$. This model has been trained for 400 epochs with a batch size of 64. As it takes a long time to train the network, it is difficult to use a *K-Fold CV* technique. The following results have been achieved using a classical approach of *test/train split* (1/3 - 2/3).

RESULTS FOR BUSH

Sample	F1	Recall	Precision	Accuracy
Train set	0.9016	1.0	0.8209	0.9912
Test set	0.7911	0.9096	0.7	0.9807

(9)

	Prev Best F1 (SVM)	New Best F1 (CNN)
Mean F1	0.6476	0.7911

(10)

RESULTS FOR WILLIAMS

Sample	F1	Recall	Precision	Accuracy
Train set	0.8974	1.0	0.8139	0.9990
Test set	0.7027	0.7647	0.65	0.9975

(11)

	Prev Best F1 (SVM)	New Best F1 (CNN)
Mean F1	0.50536	0.7027

(12)

COMMENTS

From both tables above, it is clear that the results achieved by this *CNN* are the best ones so far. The train results might suggest that the network has over-fitted, but the test results are still very good. With some additional time, it could have been interesting to actually try a *K-Fold CV* with this network to have a better estimation of the test F1.

PHASE 4

For this phase we had to select an external dataset to pre-train the network. Why ? Because as we do not have many observations from *Bush* and *Williams*, we can't use a complicated model nor train the simple one correctly. By pre-training the network on some bigger dataset, we can actually make it learn properly how to recognize useful traits to classify faces. Once this new pre-trained model is able to detect interesting features from a face picture, we can re-train it but this time on the Bush/Williams dataset so we can specialize it for our purpose. It is supposed to help the network generalize its learning and then apply it on something more specific. It should limit the overfit and help overall performances.

Which dataset to choose ? I used this one: `http://vintage.winklerbros.net/facescrub.html`. It is only constituted of faces that are finely cropped to exactly what we want. Moreover, it has a good number of observations (100,000 observations and 530 people - approx 200 pictures for each) without being too heavy either. It also has the advantage of having actual names on those faces so we can train the network on specific people instead of just using random faces.
I had very little pre-processing to do. I just had to resize and converted every image to gray-scale color. I saved each individual person into some **.npy* files to be able to get them back easily and use them for the training.
Script to download the dataset more easily: `https://github.com/faceteam/facescrub`

THE MODEL

For this new task I decided to update my network. As we now have a much bigger set of images, it is possible to improve the model. I added two convolutional layers as well as some dropouts and two final dense layers. I also increased the number of filters per convolutional layer. The activation functions and the final sigmoid layer remain untouched.

```
   OPERATION           DATA DIMENSIONS   WEIGHTS(N)   WEIGHTS(%)

       Input   #####     64   64    1
      Conv2D    \|/  -------------------       320     0.4%
        tanh   #####     64   64   32
MaxPooling2D   Y max -------------------         0     0.0%
               #####     32   32   32
      Conv2D    \|/  -------------------      9248    10.8%
        tanh   #####     32   32   32
MaxPooling2D   Y max -------------------         0     0.0%
               #####     16   16   32
     Dropout    | || -------------------         0     0.0%
               #####     16   16   32
      Conv2D    \|/  -------------------      9248    10.8%
        tanh   #####     16   16   32
MaxPooling2D   Y max -------------------         0     0.0%
               #####      8    8   32
      Conv2D    \|/  -------------------     18496    21.6%
        tanh   #####      8    8   64
MaxPooling2D   Y max -------------------         0     0.0%
               #####      4    4   64
     Dropout    | || -------------------         0     0.0%
               #####      4    4   64
      Conv2D    \|/  -------------------     36928    43.2%
        tanh   #####      4    4   64
MaxPooling2D   Y max -------------------         0     0.0%
               #####      2    2   64
     Flatten   ||||| -------------------         0     0.0%
               #####        256
     Dropout    | || -------------------         0     0.0%
               #####        256
       Dense   XXXXX -------------------     10280    12.0%
        tanh   #####         40
       Dense   XXXXX -------------------       820     1.0%
        tanh   #####         20
     Dropout    | || -------------------         0     0.0%
               #####         20
       Dense   XXXXX -------------------       210     0.2%
        tanh   #####         10
       Dense   XXXXX -------------------        11     0.0%
     sigmoid   #####          1
```

Figure 4.10: Phase 4: model architecture

The training

For Bush, I pre-trained the network on some male celebrities. I chose a good enough number of positive class picture so that it can simulate the unbalanced dataset for which we truly are interested in its performances (Bush/Williams). I used the same method for Williams, I trained the network from scratch by using female celebrities as positive class. I tried to respect the classes proportions as well. The pre-trained model has been trained for approximately 600 epochs and a batch size of 128.

Once I had the two pre-trained models (Bush/Williams) I could start the actual training for *Bush* and *Williams*. I loaded the pre-trained models and re-trained them with the same weights as in Phase 3 for approximately 100 epochs.

Results for Bush

Sample	F1	Recall	Precision	Accuracy
Train set	1.0	1.0	1.0	1.0
Test set	0.8722	0.8870	0.8579	0.9895

(13)

	Prev Best F1 (old CNN)	New Best F1 (pre-trained CNN)
Mean F1	0.7911	0.8722

(14)

Results for Williams

Sample	F1	Recall	Precision	Accuracy
Train set	1.0	1.0	1.0	1.0
Test set	0.7586	0.6470	0.9166	0.9984

(15)

	Prev Best F1 (old CNN)	New Best F1 (pre-trained CNN)
Mean F1	0.7027	0.7586

(16)

Comments

It seems pretty clear that this new method improved our final *F1-score*. However, it is difficult to tell if this is due to the network's new architecture or the pre-training phase. Either way, by looking at the training scores, it might be possible that the network overfit. I would have liked to try to oversample *Bush* and *Williams* images with some added modifications (rotations, noise, blur ...) to see if it would bring any improvements for our test scores.

Conclusion

Through this project I have been able to understand and test different parameters for *KNN*, *SVMs* and *Deep CNN*. It would appear that *KNN* is not well suited for image recognition whereas *SVMs* performed surprisingly well with and without *PCA* transformation. But as we could have expected, even though it was the most time consuming, the *Deep CNN* was the one who performed the best in terms of *F1-score*. The transfer learning seemed to work pretty well even though I am not quite sure if the improvement actually comes from the network or the pre-training.