Support Vector Machines (SVMs)

I- Description of our real classification problem

1) The dataset

In order to get real data, we have decided to use datasets from the UC Irvine Machine Learning Repository. *(http://archive.ics.uci.edu/ml/index.php)*
Then, after having filtered the datasets to get only the ones with classification task, we finally decided to choose the "Diabetic Retinopathy Debrecen Data Set".

As the abstract mention, this dataset contains **1151 samples** and **20 features** extracted from the Messidor image set to predict whether an image contains signs of diabetic retinopathy or not. The retinopathy is a complication of Diabetes.

Attributes information

1	2	3-8	9-16
The binary result of quality assessment. 0 = bad quality 1 = sufficient quality.	The binary result of pre-screening, where 1 indicates severe retinal abnormality and 0 its lack.	The results of MA detection. Each feature value stands for the number of MAs found at the confidence levels alpha = 0.5, …1, respectively.	It contains the same information as 2-7) for exudates. However, as exudates are represented by a set of points rather than the number of pixels constructing the lesions, these features are normalized by dividing the number of lesions with the diameter of the ROI to compensate different image sizes.
17	**18**	**19**	**20 (output)**
The Euclidean distance of the center of the macula and the center of the optic disc to provide important information regarding the patient's condition. This feature is also normalized with the diameter of the ROI.	The diameter of the optic disc.	The binary result of the AM/FM-based classification.	Class label. 1 = contains signs of DR (Accumulative label for the Messidor classes 1, 2, 3), 0 = no signs of DR.

Source: UCI Machine Learning Repository

2) The context & problem statement

According to the National Eye Institute of the United State of America (NEI):

- Diabetic retinopathy is the most common cause of vision loss among people with diabetes and a leading cause of blindness among working-age adults.
- Between 40 and 45 percent of Americans diagnosed with diabetes have some stage of diabetic retinopathy, although only about half are aware of it.

The problem statement is to classify the patient retina as being diabetic or not diabetic taking into consideration the available features.

II- Methods

In order to proceed to the **binary classification**, we have decided to use the **SVM** method with **sklearn**.

First of all, we began by downloading our file as a numpy matrix.
Then, because features and class are in the same file, we needed to separate the data from the target attributes. So, X takes the features, and y the class (ouput).

```
dataset = np.loadtxt("data.txt", delimiter=",")

X=dataset[0:920,0:19]
y=dataset[0:920,19:]
```

The goal is to find predictive relationships from data. To do so, we worked with a **training set** for discovering relationship and **testing set** for evaluating whether the discovered relationships hold. The training set counts **920** samples (80% of dataset) whereas the test set counts **231** samples (20%).

```
X=dataset[0:920,0:19]
y=dataset[0:920,19:]
X_test=dataset[920:,0:19]
y_test=dataset[920:,19:]
```

At the beginning, we could not even know which one of the kernels gives better model for our data set. It is why we have decided to implement 3 of them: **linear**, **polynomial**, **rbf** and then measure the model performance for each in order to know whether our model fits better with a linear, polynomial or rbf SVM.
For each kernel, we fit the SVM model according to the given training data using the fit() function.

```
clf.fit(X,y.ravel())
```

As scikit-learn.org mention, the **GridSearchCV** instance implements the usual estimator API: when "fitting" it on a dataset all the possible combinations of parameter values are evaluated and the **best combination is retained**. So, in our case, we will use it in order to estimate the best kernel with the best C and gamma parameters.
This will help us to figure out which model is the best.

```
parameter_candidates = [
  {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
  {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
    {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['poly']},
]

clf4 = GridSearchCV(estimator=svm.SVC(), param_grid=parameter_candidates, n_jobs=-1)
clf4.fit(X_train,y_train.ravel())


# View the accuracy score
print('Best score for X:', clf4.best_score_)

# View the best parameters for the model found using grid search
print('Best C:',clf4.best_estimator_.C)
print('Best Kernel:',clf4.best_estimator_.kernel)
print('Best Gamma:',clf4.best_estimator_.gamma)
```

Despite the fact that we already knew the best kernel (and its parameters) for our model, we have decided to change the value of C, which is a variable that represents **penalty parameter of the error term**, in order to check the variations. Finally, we have noticed that this parameter can influence the **over-fitting/under-fitting** of our model.
We will discuss more about that in the section « Discussion on the results ».

Note: Because we are dealing with 19 features, we did not represent our model in a visual graphic. However, the model performance measurements give us a feedback of the accuracy.

III- Results

```
Best score for X: 0.7521739130434782
Best C: 100
Best Kernel: linear
Best Gamma: auto
```

Figure 2 : GridSearchCV for training set

```
Best score for X: 0.7142857142857143
Best C: 100
Best Kernel: linear
Best Gamma: auto
```

Figure 1 : GridSearchCV for test set

	Linear	Poly	RBF
Training	0.769565217391	0.691304347826	0.703260869565
Test	0.744588744589	0.722943722944	0.718614718615
C	10	1	1
Gamma	auto	0.00010	0.00010

Accuracy of our model depending of the kernels (test set)

Note: In our script, we have decreased the value of parameter C of the linear kernel in order to make you gain some time in the script's running.

IV- Discussion on the results

We got approximately the same accuracy for our 3 kernels. The best one is kernel with almost 77% of accuracy in the test set, which is not so bad. For each kernel, our model **does not neither over-fit nor under-fit**, However, we needed to **change some parameters** in order to optimize the accuracy for each one.

In a SVM we are looking for two things:

- a hyperplane with the largest minimum margin
- a hyperplane that correctly separates as many instances as possible.

The problem is that we will not always be able to get both things. The C parameter tells the SVM optimization how much we want to avoid misclassifying each training example.
So, if we choose a large value of C, then the model will choose a smaller margin hyperplane. At the opposite, if C is small, then the margin will be large but the model will misclassify more points.
Finally, the goal was to find the **right balance between these two concepts**.

We have noticed that using a small value for gamma (gamma=0.00000001) for polynomial and rbf kernels **made the model to be more inclined to under-fitting**. Indeed, gamma defines *« how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close' »*[1]. So more the gamma is low, and more the decision boundary is linear, a little bit smoother and a little bit less jagged.

In addition, we realized that **the models over-fitted** when we put a huge value of c for rbf kernel. It Is not surprising because the decision boundary is almost entirely dependent on individual points, creating what we call "islands". The accuracy was about 95% for training set against only 60% for testing set.
At the opposite, when we put a small value for C and gamma, our model under-fitted because it could not neither model the training data nor generalize to new data.
So, we concluded that it is really important to find perfect values of C and gamma, because some high values for C and gamma can lead to overfitting, and small values can lead to under fitting.

Our model is correct thanks to the parameters we used. Indeed, as mention in the Method part, using **GridsearchCV** helped us **to optimize our classifier by choosing the right parameters**. However, we can even do better by telling GridSearchCV to check more values (for example, in our script we only asked to check for c=1,10,100,1000), but the **cost of time** would be **enormous**. Because we have done this for our 3 kernels, the running of the script ended up after more than **25 minutes**. That's why we have commented its section in our script, but if you want to test it, feel free to uncomment it.

Moreover, we can underline that SVMs are **sensitive to features scales**. In our example, we have an attribute starting from 0 as minimum and 1 as maximum, and another attribute that has 50 as maximum. This can lead to an inefficient model.

[1] Source : http://scikit-learn.org

The solution could be to use **Scikit-Learn's StandardsScaler**. In our case, even after having scaled the data, the accuracy was approximately the same.

To conclude, we were curious to compare the SVM solution with **RandomForest** that we learnt last year. It seems that random Forest is intrinsically suited for multiclass problems, while SVM is intrinsically two-class, so we wanted to check that:

```
from sklearn.ensemble import RandomForestClassifier

clf5 = RandomForestClassifier(n_estimators=10)
clf5.fit(X_train, y_train)
print("Training set with Random forest : %s" % clf5.score(X_train,y_train))
print("Testing set with Random forest : %s" % clf5.score(X_test,y_test))
```

Result: Model was overfitting, with 99% for training set and only 65% for testing set. After having scaled the data, the result is still the same.
So, with our data and its distribution, SVM is **a good alternative solution.**