Assignment 4

Elena Pérez-Ródenas Martínez
REDID: 827222533

October 2021

I, Elena Pérez-Ródenas Martínez, guarantee that this homework is my independent work and I have never copied any part from other resources. Also, I acknowledge and agree with the plagiarism penalty specified in the course syllabus.

1. Exercise

1.1.

Support Vector Classifiers are not good when the data has some points in the middle and the others in both extremes.

So the main ideas behind Support Vector Machines are:
1) Start with data in a relatively low dimension (i.e. 1D)
2) Move the data into a higher dimension (i.e. 2D)
3) Find a Support Vector Classifier that separates the higher dimensional data into two groups.

In Support Vector Machines we can have, for example, the data in the x-axis and in the y-axis each data squared. Since each observation has x and y-axis coordinates, the data are 2-Dimensional. And from there we can use a Support Vector Classifier to classify the observations.

When the data are 1-Dimensional a Support Vector Classifier is a point; in 2-Dimensions it's is a line and in 3-Dimensions it's a plane.

1.2.

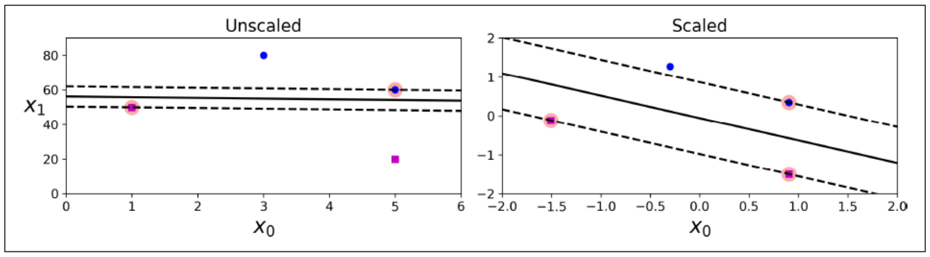
Maximal Margin Classifiers are super sensitive to outliers in the training data and that makes them not good enough. To make a threshold that is not so sensitive to outliers we must allow misclassifications. Before allowing misclassifications we pick a threshold that is very sensitive to the training data (low bias) and it performs poorly when we have new data (high variance). In contrast, when we pick a threshold that is less sensitive to the training data and allow misclassifications (higher bias) if performs better when we have new data (low variance).

We use Cross Validation to determine how many misclassifications and observations to allow inside of the soft margin to get the best classification.

When we use a Soft Margin to determine the location of a threshold we are using a Support Vector Classifier to classify the observations. The observations on the edge and within the Soft Margin are called **Support Vectors**.

1.3.

If we have two vectors X0 and X1, the first one with range (0 to 6) and the other (0 to 80) the Support Vector Machine classifier will be a linear boundary in the X0-X1 plane. However, the slope of that boundary should not depend on the range of X1 and X2 but on the distribution of the points. The problem is that if we make a prediction on the point (0.5, 60) and (1.5, 60) the value of the function will be practically the same and SVM will be less accurate since it will have less sensitivity to points in the X0 direction. If we scale it though, SVM will be more accurate because the sensitivity in both axis will be the same.



As we can see in the picture above, before scaling only two points are in the decision boundary meanwhile in the scaled picture there are three. This is a sign to see that it is always better to scale when using SVM.

1.4.

An SVM classifier can output the distance between the test instance and the decision boundary, and you can use this as a confidence score. However, this score cannot be directly converted into an estimation of the class probability. If you set probability=True when creating an SVM in Scikit-Learn, then after training it will calibrate the probabilities using Logistic Regression on the SVM's scores (trained by an additional five-fold cross-validation on the training data). This will add the predict_proba() and predict_log_proba() methods to the SVM. [1]

1.5.

When our data is not separable we introduce a penalty:
$min_{w,b,\xi_i \geq 0} \frac{1}{2}|w|^2 + C\sum_{i=1}^{n} \xi_i$ s.t. $\forall i$, $y_i(w \cdot x_i + b) \geq 1 - \xi_i$

We want to minimize $|w|^2$ plus the number of training mistakes and we will set C using cross validation.

Hard-margin SVM: Try to find a hyperplane that best separates positive from negative points, such that no point is misclassified.

Soft-Margin SVM: Try to find a hyperplane that best separates positive from negative points, but allows for some points to be misclassified, in which case the objective function is punished proportionally to degree of misclassification. [2]

The C parameter controls how much you want to punish your model for each misclassified point for a given curve. When C is large, larger slacks penalize the objective function of SVM's more than when C is small so the lower the C parameters, the softer the margin. As C approaches infinity, this means that having any slack variable set to non-zero would have infinite penalty. Consequently, as C approaches infinity, all slack variables are set to 0 and we end up with a hard-margin SVM classifier.

On the other hand when C is small and approaches 0, our slack variables for all data points are free to be as large as possible to maximize the margin and it's easy for our model to underfit the training data. [3]

1.6.

One way to deal with overlapping data is to use a Support Vector Machine with a Radial Kernel (Radial Basis Function, RBF). It behaves like a Wighted Nearest Neighbor model, i.e., the closest observations have a lot influence on how we classify the new observation and observations that are further away have relatively little influence on the classification.

We calculate $e^{-\gamma(a-b)^2}$ where a and b are two different observations, and γ is determined by Cross Validation and scales the squared distance and its influence.

By scaling the distance, γ scales the amount of influence two points have on each other. The further two observations are from each other, the less influence they have on each other. So the higher the gamma, the more influence the feature data points will have on the decision boundary and the model gets overfits.

On the other hand, As the value of C increases the model gets overfits as we already saw in exercise 5 because the slack variables will be set closer to zero.

So to overfit the training set we should increase both C and γ.

2. Coding Project: Linear SVM classifier using Gradient Descent

In this coding exercise we just have to complete the code with the guidelines explained. In order to do that we will complete the function fit.

First, we will create a vector called y_ with a -1 if y<=0 or a 1 otherwise. We will save the shape of X which will be the number of samples and features.

```
y_ = np.where(y<=0, -1, 1)
n_samples, n_features = X.shape
```

After that we will initialize w and b to random numbers.

```
w = np.random.rand(n_features)
b = np.random.rand()
```

Now we have to create a container to save the value of the Js in each epoch so we will initialize it to a vector of zeros.

```
self.Js=[0 for i in range(self.n_epochs)]
```

We will set lambda to a small value like 0.01 and we will proceed to train the model. We will go through each epoch and first see if we meet the condition $y_i^{true} * (\sum_{j=1}^{n} w_j * x_j^i + b) < 1$

```
dist = y_ * (np.sum(w * X, axis=1) + b)
condition = 1 > dist
```

In order to apply gradient descent we will save the vectors which meet the condition which are the ones that are wrongly classified.

```
y_wrong = y_[condition]
X_wrong = X[condition]
```

In that case, when we compute the cost function $J(w)$, the $max(0, 1 - y_i^{true} * (\sum_{j=1}^{n} w_j * x_j^i + b))$ will be the second value and our cost function will be $J(w) = C \cdot \sum_{i=1}^{m}(1 - y_i^{true} * (\sum_{j=1}^{n} w_j * x_j^i + b))) + \lambda \sum_{j=1}^{n} w_j^2$ So we save the Js of the misclassified values.

```
self.Js[epoch] = self.C * (1 - dist[condition]).sum() + lamda * norm(w, 2)
```

The derivative of the cost function with respect to w_j will be $\frac{\partial J(w)}{\partial w_j} = 2 * \lambda * w_j + C \cdot \sum_{i=1}^{m} -y_i^{true} * x_j^i$. So we will save the gradient vector of the misclassified values.

```
w_gradient_vector = 2 * lamda * w - self.C * np.dot(y_wrong, X_wrong)
```

Also, the derivative with respect to b will be $C \cdot \sum_{i=1}^{m} -y_i^{true}$

```
b_derivative = self.C * -y_wrong.sum()
```

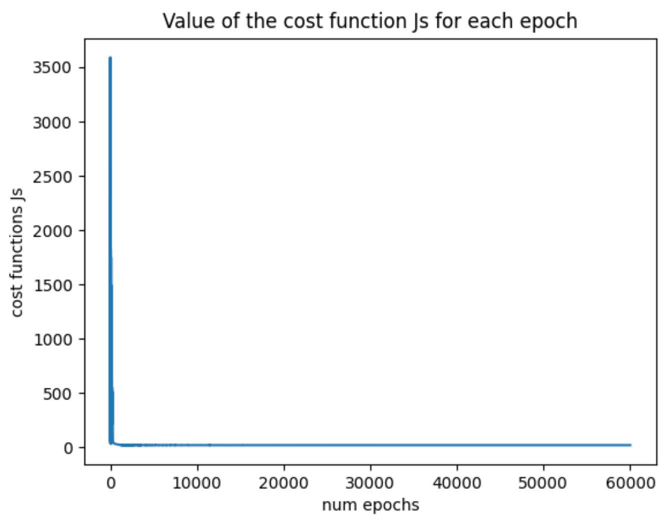
Finally, we will update w and b:

```
w = w - self.eta(epoch) * w_gradient_vector
b = b - self.eta(epoch) * b_derivative
```

Now we will plot the value of the cost function Js for each epoch in a graph. We will create a function for it called plotJs with the epochs in the x-axis and the Js in the y-axis.

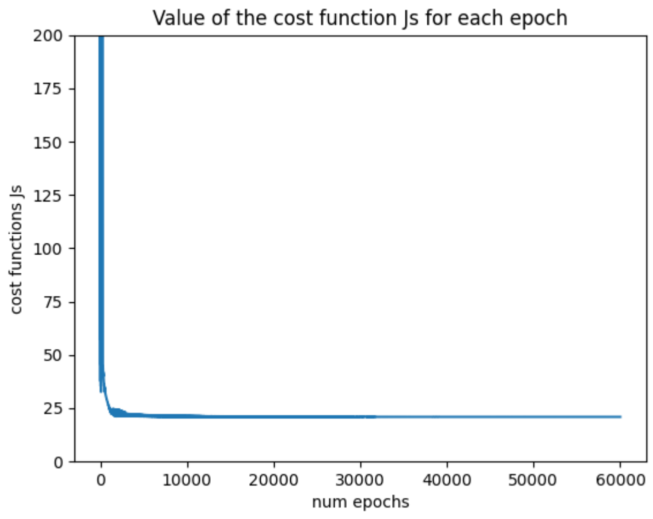
```
def plotJs(self):
    plt.plot(range(self.n_epochs), self.Js)
    plt.title("Value of the cost function Js for each epoch")
    plt.xlabel("num epochs")
    plt.ylabel("cost functions Js")
    plt.show()
```

We obtain the following graph:



At first glance, it seems like if it establishes at 0 but if we get closer setting a limit of 200 in the y-axis we can see that it actually establishes at 24 approximately.

```
plt.ylim(0, 200)
```



3. Referencias

[1] https://github.com/kalperen/MachineLearningGuide.

[2] https://queirozf.com/entries/choosing-c-hyperparameter-for-svm-classifiers-examples-with-scikit-learn.

[3] https://www.quora.com/Using-SVM-Classifier-with-C-0-and-C-infinity-what-would-be-the-effect-on-classifying-this-data.