Data Science and Economics

Machine learning, statistical learning, deep learning and artificial intelligence

News categorization with Pegasos

Andrea Joseph Froiio

922907

Index

1 Introduction **1**

2 Data and methods **2**
2.1 Dataset 2
2.2 Online Gradient Descent 2
2.3 Support Vector Machines 3
2.4 Pegasos 3

3 Prediction **5**
3.1 Parameters tuning 5
3.2 Label 33 6
3.3 Label 70 7
3.4 Label 101 8
3.5 Label 4 9

4 Conclusions **10**

Chapter 1

Introduction

The purpose of this paper is to study the effects of the parameters λ and T of the Pegasos algorithm and to find which of their combinations are best to predict the presence of a label for each of the four most frequent topics of the news in the dataset.
External cross-validation is used to evaluate accuracy.
A subset of about 20000 items from the Reuters dataset is used.
The algorithm divides the space of the datapoints and classifies them based on which of the half-spaces they fall in (positive and negative) and the output vector is ideally the one misclassifying the least number of datapoints (data may not be linearly separable).
The best combinations of λ and T are found to be almost always the same for the 4 labels.
Chapter 2 describes the tools which have been used, such as the dataset and algorithms that help to understand Pegasos better, while chapter 3 analyzes the practical work that has been done: the tuning of the parameters and the output for the 4 labels.
Eventually, chapter 4 draws some conclusions about this study.

Chapter 2

Data and methods

2.1 Dataset

The *Reuters dataset* has a row for each news, in which *doc******* is its ID, the number-labels before the "=" represent the topics of the news, sequentially there are numbers after the "=" representing the words appearing in it, with their respective frequency in the text.
For an example, this is part of the first row:

doc002286 93 59 101 92 34 = 372:0.0322298670393407 412:0.0399666607009284...

The first thing to be done is cleaning the dataset by removing the docIDs and the rows without labels, replacing the spaces between "words" with commas and ending up with a list for labels and another one for the features, which are use to recreate the dataset with 19860 clean rows. The next thing done is loading the dataset in a sparse matrix, which allows to save space by storing only non-zero values.

2.2 Online Gradient Descent

Online learning is a learning protocol for algorithms which work on a continuous flow of new data. Differently than conventional algorithms for machine learning, an "online learning" algorithm is not trained from zero each time, but instead is tweaked as new data comes in.
The *statistical risk* is the evaluation of the performance of a predictor $h : \mathcal{X} \to \mathcal{Y}$ with respect to $(\mathcal{D}, \ell)$, where ℓ is the loss function and $\mathcal{D}$ is the unknown probability distribution on $\mathcal{X} \times \mathcal{Y}$, and it is defined by: $\ell_{\mathcal{D}}(h) = \mathbb{E}[\ell(Y, h(\boldsymbol{X}))]$, that is the expected value of the loss function on the random sample.
Since there is a sequence of loss functions, now instead of the statistical risk, *sequential risk* is used to evaluate the accuracy of the algorithm:

$$\frac{1}{T}\sum_{t=1}^{T} \ell(h_t(x_t), y_t) \tag{2.1}$$

The *regret*, which will be used later, is the difference between the sequential risk of $h_1, ...h_t$ and the sequential risk of the best predictor in the class $\mathcal{H}$ (which is the model space), for the loss functions $\ell_1, ...\ell_t$:

$$\frac{1}{T}\sum_{t=1}^{T} \ell_t(h_t) - \min_{h\in\mathcal{H}} \frac{1}{T}\sum_{t=1}^{T} \ell_t(h) \tag{2.2}$$

Online Gradient Descent (OGD) is a local optimization method used to locate stationary points that are suitable for convex optimizations and it is the "online" version of gradient descent.

Below a pseudo-code for the OGD algorithm:

Algorithm 1: OGD algorithm for λ-strongly convex functions

Initialize $w_1 = 0$
for $t = 1, 2...$ **do**
  1) $w_{t+1} = w_t - \frac{1}{\lambda t}\nabla \ell_t(w_t)$
end

where $\ell_1, \ell_2...$ are all λ-strongly convex losses.
The final result is:

$$\frac{1}{T}\sum_{t=1}^{T}\ell_t(w_t) \leq \min_{u\in\mathbb{R}^d}\frac{1}{T}\sum_{t=1}^{T}\ell_t(u) + \frac{G^2}{2\lambda}\frac{ln(T+1)}{T} \tag{2.3}$$

where is chosen $G \geq \max_t ||\nabla \ell_t(w_t)||$.

2.3 Support Vector Machines

Support Vector Machines (SVM) is a classification technique that splits the data in "the best possible way", which means that it finds the farthest hyperplane w^* from the *support vectors* by maximizing the margin between them, making it the one with the least probability of misclassification.
Support Vectors are the closest positive and negative x_t to the hyperplane, on which it has margin equal to 1: $y_t(w^*)^T x_t = 1$.
Given a linearly separable training set $(x_1, y_1), ..., (x_m, y_m) \in \mathbb{R}^d \times \{-1, +1\}$ to maximize the margin, you must solve a constrained optimization problem

$$\min_{w\in\mathbb{R}^d}\frac{1}{2}||w||^2 \tag{2.4}$$

$$\text{s.t. } y_t w^T x_t \geq 1$$
$$t = 1, ..., m$$

Where

$$w^* = \sum_{t\in I}\alpha_t y_t x_t \tag{2.5}$$

knowing that, if w_0 is an optimal solution, $\alpha \in \mathbb{R}^m$ is a vector such that $\nabla f(w_0) + \sum_{t\in I}\alpha_t \nabla g_t(w_0) = 0$ in which $I = \{1 \leq t \leq m : g_t(w_0) = 0\}$ and g_t is a differentiable function.
Since data is rarely linearly separable, the SVM objective function can be generalized for non-linearly separable cases:

$$w^* = \underset{w\in\mathbb{R}^d}{\text{argmin}}\left(\frac{1}{m}\sum_{t=1}^{m}\xi_t + \frac{\lambda}{2}||w||^2\right) \tag{2.6}$$

Where $\xi_t \geq 0$ is the *slack* and λ is the *regularization* parameter introduced in order to balance the two.
The slack variable ξ_t measures the violation of each *support vector* by the potential solution w.
To minimize it: $\xi_t = [1 - y_t w^T x_t]_+ = h_t(w)$, which is the *hinge loss*, so the problem can be rewritten as

$$w^* = \underset{w\in\mathbb{R}^d}{\text{argmin}}\left(\frac{1}{m}\sum_{t=1}^{m}h_t(w) + \frac{\lambda}{2}||w||^2\right) \tag{2.7}$$

2.4 Pegasos

Pegasos is a particular instance of OGD for strongly convex functions, and it is used to solve the minimization problem of SVM.

$$F(w) = \frac{1}{m}\sum_{t=1}^{m}h_t(w) + \frac{\lambda}{2}||w||^2 = \frac{1}{m}\sum_{t=1}^{m}\ell_t(w) \tag{2.8}$$

where $\ell_t(w)$ is the λ-strongly convex hinge loss.
It consists of the following algorithm:

Algorithm 2: Pegasos

Parameters : number of T iterations, coefficient λ of regularization
Input : Training set $(x_1, y_1), ..., (x_m, y_m) \in \mathbb{R}^d \times \{-1, +1\}$
Initialize $w_1 = 0$
for $t = 1, ..., T$ **do**
  1) Randomly extract uniformly an element (x_{Z_t}, y_{Z_t}) of the training set
  2) $w_{t+1} = w_t - \eta_t \nabla \ell_{Z_t}(w_t)$
end
Output : $\overline{w} = \frac{1}{T}(w_1 + ... + w_T)$

Given w^* as the solution to the SVM objective function, the OGD analysis for the strongly convex functions gives

$$\frac{1}{T}\sum_{t=1}^{T} \ell_{s_t}(w_t) \leq \frac{1}{T}\sum_{t=1}^{T} \ell_{s_t}(w^*) + \frac{G^2}{2\lambda T} ln(T+1) \tag{2.9}$$

where $G = \max_{t=1,...,T} ||\nabla \ell_{s_t}(w_t)||$ is a Random Variable.
This can be used to upper bound $F(\overline{w})$:

$$\mathbb{E}[F(\overline{w})] \leq F(w^*) + \frac{\mathbb{E}[G^2]}{2\lambda T} ln(T+1) \tag{2.10}$$

So $\overline{w}$ converges to w^* at rate $\frac{lnT}{T}$.
To upper bound G, $||w_t||$ is upper bounded and is set $v_s = y_{s_t} x_{s_t} \mathbb{I}\{h_{s_t}(w_t) > 0\}$ and then the update of w_t is

$$w_{t+1} = \frac{1}{\lambda T}\sum_{s=1}^{t} v_s \tag{2.11}$$

So in the end

$$||\nabla \ell_t(w_t)|| \leq X + \lambda ||w_t|| \leq 2X \tag{2.12}$$

where $X \leq \max_s ||v_s||$ and replacing the upper bound in G:

$$\mathbb{E}[F(\overline{w})] \leq F(w^*) + \frac{2X^2}{\lambda T} ln(T+1) \tag{2.13}$$

Chapter 3

Prediction

The goal is to train four binary classifiers, each to recognize the presence of one of the four most frequent labels in the dataset.
The dataset contains 100 labels, going from *0* to *101*, missing *49* and *66*, probably due to its cleaning:

label	**33**	**70**	**101**	**4**	...	Total
absolute frequency	8899	5634	4874	3465	...	61876
relative frequency	0.143820	0.091053	0.078770	0.055999	...	1

We now create 4 new label lists in which, for each row, we substitute the labels with $+1$ if the label of interest for that list appears in that row, -1 otherwise:

$$y_i = \begin{cases} +1 \text{ if the i-th row has the label} \\ -1 \text{ otherwise} \end{cases}$$

Before running the algorithm, the data are split into training and test sets, respectively in 0.6 and 0.4 proportion.
In order to have a split with the same proportion of classes in the train and test sets, data are stratified, which is usually done in unbalanced dataset and makes it possible to compare training and test set results as the training will be done equally on each class present.

3.1 Parameters tuning

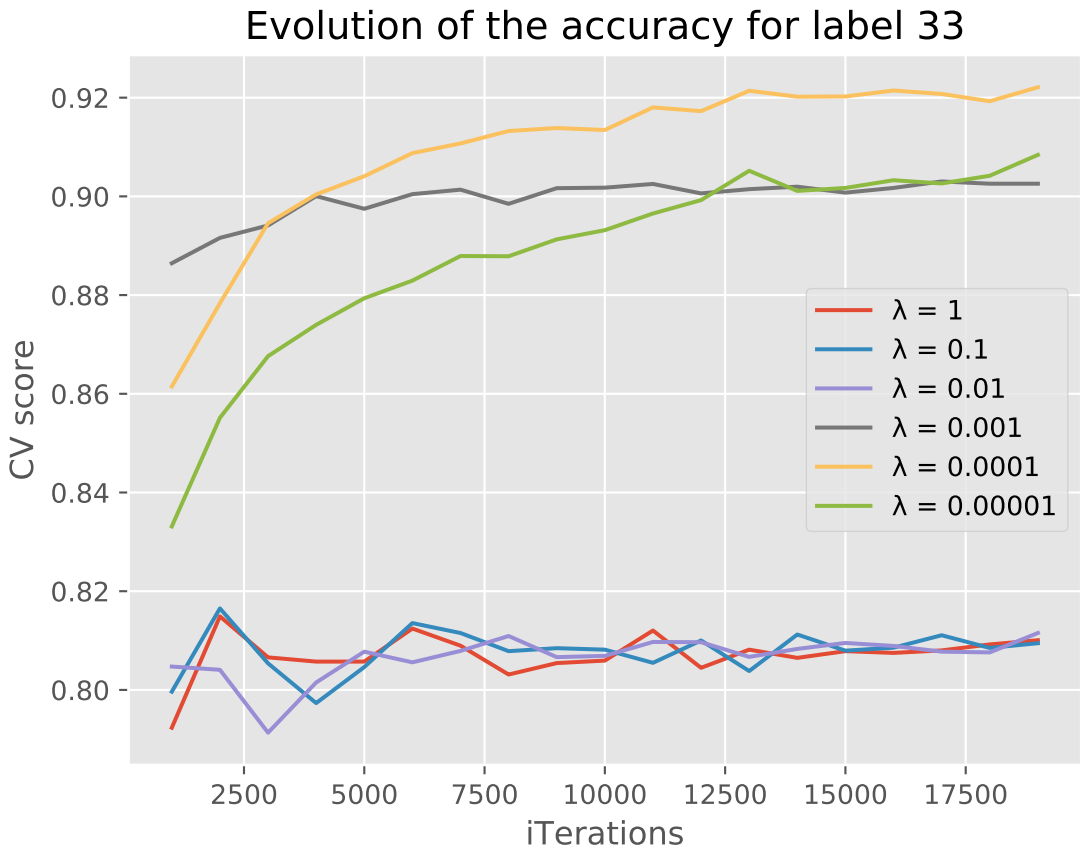
The number of iterations are set to go from 1000 to almost the cardinality of the dataset, 19000, and stepping every 1000, having 19 Ts.
λs are set to go from 1 to 0.00001, each with a decimal more than the previous one, making 6 λs.
The algorithm, for each label, will compute $19 \times 6 = 114$ combinations.

T iterations	(1000, 19001, 1000)					
λ	1	0.1	0.01	0.001	0.0001	0.00001

10 fold Cross Validation will be used to evaluate the accuracy.

3.2 Label 33

Now the algorithm is run and it plots the results for better understanding of the evolution of the classifier accuracy:



It can be clearly seen from the plot that the best λ is 0.0001, and from the dataframe of the results:

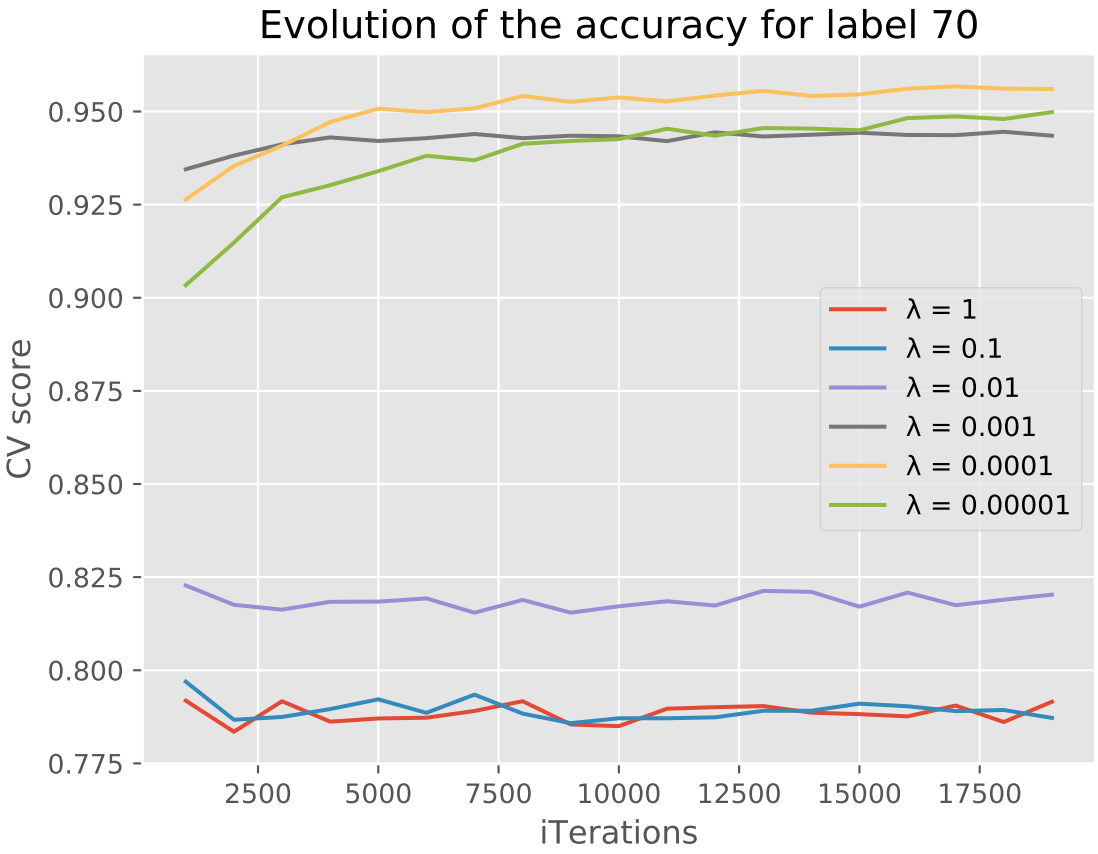
Accuracy for label **33**	
avg	0.853124
min	0.791339
max	0.922105

The maximum accuracy is reached by this combination:

T	λ	Training score	Test score	CV score
19000	0.0001	0.960977	0.921073	0.922105

3.3 Label 70

The same is done for the other labels:



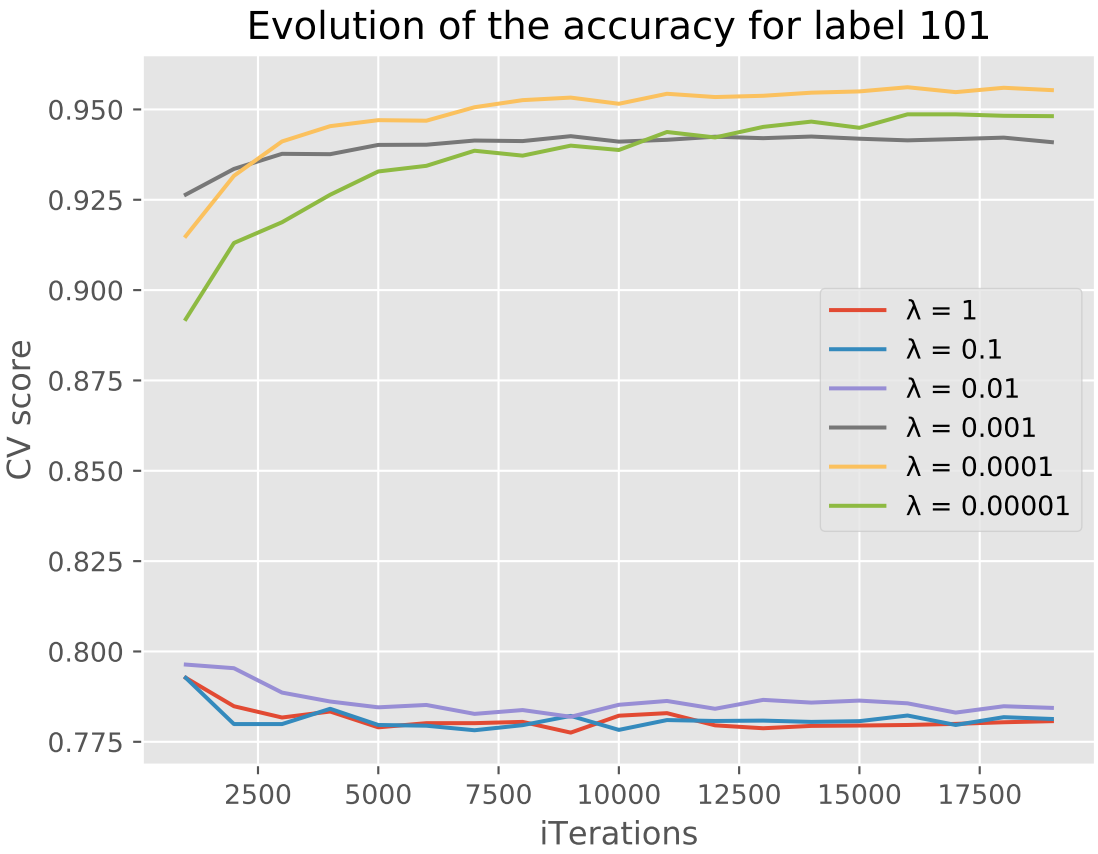
The best λ overall is again 0.0001.

Accuracy for label **70**	
avg	0.871288
min	0.783484
max	0.956747

The maximum accuracy is reached by this combination:

T	λ	Training score	Test score	CV score
17000	0.0001	0.978432	0.956193	0.956747

3.4 Label 101



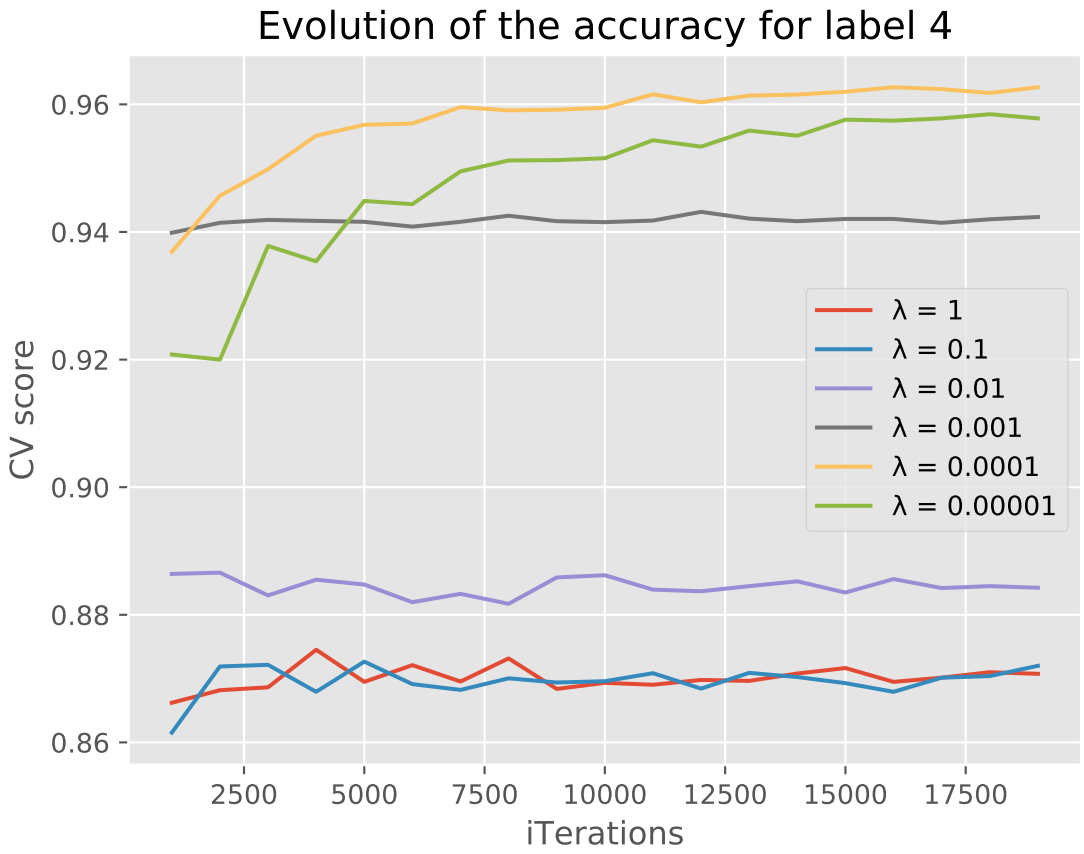
The best λ overall is still 0.0001.

Accuracy for label **101**	
avg	0.862271
min	0.777543
max	0.956143

The maximum accuracy is reached by this combination:

T	λ	Training score	Test score	CV score
16000	0.0001	0.973817	0.956445	0.956143

3.5 Label 4



Yet again the best λ is 0.0001, and from the dataframe of the results:

Accuracy for label **4**	
avg	0.911948
min	0.861531
max	0.962689

The maximum accuracy is reached by 2 combinations:

T	λ	Training score	Test score	CV score
16000	0.0001	0.978348	0.961103	0.962689
19000	0.0001	0.980195	0.960222	0.962689

Since the training and test scores are very similar, the combination in which there are fewer iterations is preferred, in this case 16000.

Chapter 4

Conclusions

label	**33**	**70**	**101**	**4**
optimal T	19000	17000	16000	16000
optimal λ	0.0001	0.0001	0.0001	0.0001
CV accuracy	0.922105	0.956747	0.956143	0.962689

From the plots it is visible that, overall, the best λs are the 3 smaller ones, but the best isn't the smallest one.
Therefore a lesson learned, other than the bigger λs are much worse, is that not to exceed in reducing λ, and in this case the best one is 0.0001.
This is due to the task of the regularization coefficient, which must find a balance for the slack, allowing the hyperplane to misclassify some of the datapoints in the dataset, that may not be linearly separable.
It is notable that in the 3 best λs, as T grows, the accuracy line increases steeply until about 10000, then it becomes "flat" , rendering almost useless any further increase in T.
If one is not "budgeting" T, the best accuracy scores are reached for iterations that go from 16000 to 19000 and are all spanning from about 0.92 to 0.96, therefore well above 90% accuracy, which is good.