Introduction

Consider the classification problem for two non-intersecting classes, in which objects are described by n-dimensional real vectors: $X = R^n, Y = \{-1, +1\}$.

Will build a linear threshold classifier:

$$a(x) = sign\left(\sum_{j=1}^{n} w_j x_j + b\right) = sign(< w, x > +b), \qquad (1)$$

where $x = (x_1, \dots, x_n)$-featured description of object x; vector $w = (w_1, \dots, w_n) \in R^n$ and scalar threshold $b \in R$ are classifier parameters. Equation $< w, x >= -b$ describes hyperplane that splits the classes in space R^n.

In case, if given set $X = (x_i, y_i)_{i=1}^{l}$ is linearly separable , then there is such w, b that error functional $Q(w, b) = \sum_{i=1}^{l}[y_i(< w, x_i > +b) < 0]$ becomes zero, but separating hyperplane location is not strictly defined by mentioned constraints and to optimally chose it SVM method generally uses the idea of margin maximization – the separating hyperplane should be as far as possible from nearest points of both classes.

Also, it is seen that classifier parameters are determined up to normalization - $a(x)$ remain the same in case of multiplying all the parameters by some positive scalar (positive value). It is convenient to choose it in such way that satisfies the next constraints: $\min_{i=1,\dots,l} y_i(< w, x_i > +b) = 1$

Set of points $\{x: -1 \leq < w, x > +b \leq 1\}$ describes lane that splits the classes. There are no objects from training set inside this lane. Lane borders are two hyperplanes with normal given by vector w. Separating hyperplane goes strictly between this borders. Nearest to the separating hyperplane objects lie on the lane borders and minimum is reached on them.

Assume two training objects $\{x_-, x_+\}$ from different classes $\{-1,1\}$ that lie on lane borders, then lane width is given by:

$$< (x_+ - x_-), \frac{w}{||w||} >= \frac{< w, x_+ > - < w, x_- >}{||w||} = \frac{(-b+1) - (-b-1)}{||w||} = \frac{2}{||w||}$$

The lane width is maximal when the norm $||w||$ minimal.

So, in case of linearly separable objects the classifier “tuning” is transforming into the quadratic programming problem (2): find w, b values that satisfy l constraints-inequalities and $||w||$ is minimal.

$$\begin{cases} < w, w > \to min \\ y_i(< w, x_i > +b) \geq 1, i = 1, \dots, l \end{cases} \qquad (2)$$

But, in real life, linearly separable objects are rare.

To generalize the problem on linearly inseparable case, let the algorithm make mistakes, but will try to make their amount as less as possible. After adding additional variable $\xi_i \geq 0$ that define the error on

objects $x_i, i = 1, \dots, l$ and updating the optimization functional the problem assumes form of the system (2') given below:

$$\begin{cases} \frac{1}{2} < w, w > + C \sum_{i=1}^{l} \xi_i \to \min\limits_{w, w_0, \xi} \\ y_i(< w, x_i > + b) \geq 1 - \xi_i, i = 1, \dots, l \\ \xi_i \geq 0, i = 1, \dots, l \end{cases} \quad (2')$$

Minimization factor $g(w, \xi_i) = \frac{1}{2} < w, w > + C \sum_{i=1}^{l} \xi_i$ may be presented in the form given in the assignment: $G(w, \xi_i) = \frac{1}{l} \sum_{i=1}^{l} \xi_i + \frac{\mu}{2} < w, w >$. Then both function will lead to the same result if: $G(w, \xi_i) = \mu \cdot g(w, \xi_i), where\ C = \frac{1}{l\mu}$. But $g(w, \xi_i)$ is more comfortable to dealing with.

Positive constant C is controlling parameter of the method that allows to find a compromise between lane borders width maximization and total error minimization.

In problems with two classes $Y = \{-1, +1\}$ the indentation of the object x_i from the class border is the next value: $M_i(w, b) = y_i(< w, x_i > + b)$.

The algorithm makes a mistake on the object x_i only when indentation M_i is negative. If $M_i \in (-1, +1)$, then object x_i lie inside the separating lane. If $M_i > 1$, then object is classified correctly and lies on some distance from separating lane.

Due to $\sum_{i=1}^{l} \xi_i$ minimization some of constraints $\xi_i \geq 0$ and $\xi_i \geq 1 - M_i$ is transformed into equality. And so, error could be defined through indentation: $\xi_i = (1 - M_i)_+$. The problem is equivalent to unconditional minimization of the functional Q, independent from ξ_i:

$$Q(w, b) = \sum_{i=1}^{l} \big(1 - M_i(w, b)\big)_+ + \frac{1}{2C} \big||w|\big|^2 \to \min_{w, w_0}$$

Will define Lagrange function (3) for given problem:

$$\begin{aligned} L(w, b, \xi; \lambda, \eta) &= \frac{1}{2} \big||w|\big|^2 + C \sum_{i=1}^{l} \xi_i - \sum_{i=1}^{l} \lambda_i (M_i(w, b) - 1 + \xi_i) - \sum_{i=1}^{l} \xi_i \eta_i \\ &= \frac{1}{2} \big||w|\big|^2 - \sum_{i=1}^{l} \lambda_i (M_i(w, b) - 1) - \sum_{i=1}^{l} \xi_i (\lambda_i + \eta_i - C), \end{aligned} \quad (3)$$

where $\lambda = (\lambda_1, \dots, \lambda_l)$ – vector of variables are dual to w; $\eta = (\eta_1, \dots, \eta_l)$ – vector of variables dual to $\xi = (\xi_1, \dots, \xi_l)$.

According to the Kuhn-Tucker theorem, the problem is equivalent to the dual problem (4) of finding the saddle point of the Lagrange function:

$$\begin{cases} L(w, b, \xi; \lambda, \eta) \to \min\limits_{w, b, \xi} \max\limits_{\lambda, \eta}; \\ \xi_i \geq 0, \lambda_i \geq 0, \eta_i \geq 0, i = 1, \dots, l; \\ \lambda_i = 0\ or\ M_i(w, b) = 1 - \xi_i, i = 1, \dots, l \\ \eta_i = 0\ or\ \xi_i = 0, i = 1, \dots, l \end{cases} \quad (4)$$

A necessary condition (5,6,7) for the saddle point of the Lagrange function is zero derivatives:

$$\frac{dL}{dw} = w - \sum_{i=1}^{l} \lambda_i y_i x_i = 0 \to w = \sum_{i=1}^{l} \lambda_i y_i x_i ; \quad (5)$$

$$\frac{dL}{db} = -\sum_{i=1}^{l} \lambda_i y_i = 0 \to \sum_{i=1}^{l} \lambda_i y_i = 0; \quad (6)$$

$$\frac{dL}{d\xi_i} = -\lambda_i - \eta_i + C = 0 \to \eta_i + \lambda_i = C, i = 1, \dots, l. \quad (7)$$

From given above, it is seen that vector w is linear combination of training vectors x_i, and only for those which $\lambda_i > 0$. If $\boldsymbol{\lambda_i > 0}$, corresponding object x_i is called as **support vector**.

Third equation and inequality $\eta_i \geq 0$ gives $0 \leq \lambda_i \leq C$. From given above it is seen that only three set of variables ξ_i, λ_i, η_i and indentations M_i are possible:

1. $\lambda_i = 0;\ \eta_i = C;\ \xi_i = 0; M_i \geq 1$
   Object x_i classified correctly and doesn't affect on solution w. Such objects will be called as **peripheral**.
2. $0 < \lambda_i < C; 0 < \eta_i < C;\ \xi_i = 0; M_i = 1.$
   Object x_i classified correctly and lies exactly on the separating lane border. Such objects will be called as **boundary support**.
3. $\lambda_i = C;\ \eta_i = 0;\ \xi_i > 0; M_i < 1.$
   Object x_i lies inside separating lane, but classified correctly ($0 < \xi_i < 1, 0 < M_i < 1$), or lies on boundaries ($\xi_i = 1, M_i = 0$) or even lies in opposite class area ($\xi_i > 1, M_i < 0$). In all these cases object x_i will be called as **supporting intruder.**

Due to relations (7) all ξ_i, η_i members in Lagrangian (3) is nullified and it is defined only through dual variables λ_i:

$$\begin{cases} -L(\lambda) = -\sum_{i=1}^{l} \lambda_i + \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} \lambda_i \lambda_j y_i y_j < x_i, x_j > \to \min\limits_{\lambda}; \\ 0 \leq \lambda_i \leq C, \qquad i = 1, \dots, l \\ \sum_{i=1}^{l} \lambda_i y_i = 0 \end{cases} \quad (8)$$

Here (8) quadratic functional minimizing. It has non-negative definite quadratic form, hence it is convex. Area defined by constraints (inequalities and one equality) is also convex, and so given problem has unique solution.

Assume that problems (8) is solved. Then, vector w is defined by expression (5). To estimate the b it is possible to use $b = y_i - < w, x_i >$ with one or few **boundary support vectors.** For better results it is recommend to use median above all boundary support vectors as given below:

$$b = med\{< w, x_i > - y_i : \lambda_i > 0, M_i = 1, i = 1, \dots, l\}. \quad (9)$$

Finally, SVM classifier takes form of expression (10):

$$a(x) = sign\left(\sum_{i=1}^{l} \lambda_i y_i < x_i, x > + b\right) \qquad (10)$$

Summation in a(x) is made only for those objects x_i where $\lambda_i \neq 0$. Classifier will remain the same if all other (then $\lambda_i \neq 0$) objects will be deleted from training set.

Dual problem (10) is a quadratic programming problem. General solving methods for this problem are known but they laborious in sense of implementation and execution time. Therefore, usually to train the SVM algorithms that take into account SVM specific features are used. The main feature is that amount of support vectors is not so great $h \ll l$, and these vector lie near the classes borders. These features helps to accelerate SVM training.

There are numerous algorithms used for solving mentioned here QP problem, but in the given work we decided to investigate two algorithms that: incremental active set method and gradient descent method (PEGASOS). Both of them should effective and simple in one time. To compare the algorithms it would be reasonable to compare quality of classification results in sense of training and tuning error and execution time.

Algorithms

Incremental active set method

One of such algorithms is “incremental active set method, INCAS”. Will rewrite the problem in matrix form. Let introduce the notation: matrix $Q = \left(y_i y_j < x_i, x_j >\right)_{i=1,l}^{j=1,l}$ of size $l \times l$, vector of classes (answer) $y = (y_i)_{i=1,l}$, vector of dual variables $\lambda = (\lambda_i)_{i=1,l}$ and unity vector $e = (1)_{i=1,l}$. Then problem assumes form of system (11):

$$\begin{cases} \frac{1}{2}\lambda^T Q\lambda - e^T\lambda \to \min\limits_{\lambda} \\ y^T\lambda = 0 \\ 0 \le \lambda \le Ce \end{cases} \qquad (11)$$

Assume that solution λ is yet unknown, but separation of the object set x on three disjoint sets $\{1, \dots, l\} = I_O \cup I_C \cup I_S$ is known:

$$\begin{aligned} I_O = \{i: \lambda_i = 0\} &\quad - \textbf{peripheral objects}, \mathrm{M_i} > 1 \\ I_S = \{i: 0 < \lambda_i < C\} &\quad - \textbf{support objects}, \mathrm{M_i} = 1 \\ I_C = \{i: \lambda_i = C\} &\quad - \textbf{intruder object}, \mathrm{M_i} < 1 \end{aligned}$$

Constraints inequalities $0 \le \lambda_i \le C$ become **active** (is transforming into equalities) on peripheral objects ($i \in I_O, \lambda_i = 0$) and intruder objects ($i \in I_C, \lambda_i = C$). After substitution of the corresponding values of λ_i in (11) the problem become dependent only from $\lambda_i, i \in I_S$. To apply this will split the matrix Q, y, e, λ as follow:

$$Q = \begin{pmatrix} Q_{SS} & Q_{SO} & Q_{SC} \\ Q_{OS} & Q_{OO} & Q_{OC} \\ Q_{CS} & Q_{CO} & Q_{CC} \end{pmatrix}; y = \begin{pmatrix} y_S \\ y_O \\ y_C \end{pmatrix}; e = \begin{pmatrix} e_S \\ e_O \\ e_C \end{pmatrix}; \lambda = \begin{pmatrix} \lambda_S \\ 0 \\ Ce_C \end{pmatrix}$$

In the given above notation the problem (11) assumes form of (11’) given below:

$$\begin{cases} \frac{1}{2}\lambda_S^T Q_{SS}\lambda_S - Ce_C^T Q_{CS}\lambda_S - e^T\lambda_S \to \min\limits_{\lambda_S} \\ y_S^T\lambda_S + Ce_C^T y_C = 0 \end{cases} \qquad (11')$$

(11’) seems to be an minimization of quadratic functional from $h = |I_S|$ variables with one equality constraint. This problem can easily be solved by linear algebra methods though inversing of the definite positive symmetric matrix Q_{SS} of size $h \times h$.

Solution of (11’) gives full λ vector that makes it possible to estimate SVM parameters w, b by expressions (5) and (9). When w, b is become known it is become possible to estimate indentations M_i and check if objects sets I_O, I_S, I_C has been defined correctly. If Kuhn-Tucker conditions are not violated on any object, then solution is found. Else, if Kuhn-Tucker conditions are violated on some object then this object should be transferred into another set:

1. IF $i \in I_S \; and \; \lambda_i \le 0$, then object x_i should be transferred from $I_S \; to \; I_O$

2. IF $i \in I_S \; and \; \lambda_i \geq C$, then object x_i should be transferred from $I_S \; to \; I_C$
3. IF $i \in I_O \; and \; M_i \leq 1$, then object x_i should be transferred from $I_O \; to \; I_S$
4. IF $i \in I_C \; and \; M_i \geq 1$, then object x_i should be transferred from $I_C \; to \; I_S$

After each such sets modifications the problem (11') should be solved again. Iterations continues until Kuhn-Tucker satisfaction on all objects.

"Incremental active set" method **pseudocode**

Input:
X - training set
C - training parameter
Output:
w, b - SVM classifier parameters

Initial assumption:
$I_S =$ two nearest points from different classes
$I_O =$ all other than I_S points
$I_C = \emptyset$
repeat
  repeat
    solving optimization problem (find λ_S):

$$\begin{cases} \frac{1}{2}\lambda_S^T Q_{SS}\lambda_S - Ce_C^T Q_{CS}\lambda_S - e^T\lambda_S \to \min\limits_{\lambda_S} \\ y_S^T\lambda_S + Ce_C^T y_C = 0 \end{cases} \quad (11')$$

    if $|I_S| > 2 \; and \; \exists i: i \in I_S \; and \; \lambda_i \leq 0$ **then** transfer $i \; to \; I_O$
    if $|I_S| > 2 \; and \; \exists i: i \in I_S \; and \; \lambda_i \geq C$ **then** transfer $i \; to \; I_C$
  while exists such $i \in I_S$ that should be transferred to I_O or I_C
  estimate w, b parameters by expressions (5), (9)
  estimate indentations $M_i, i \in I_O \cup I_C$
  if $\exists i: i \in I_O \; and \; M_i \leq 1$ then transfer $i \; to \; I_S$
  if $\exists i: i \in I_C \; and \; M_i \geq 1$ then transfer $i \; to \; I_S$
while exists such $i \in I_O \cup I_C$ that should be transferred to I_S

To deal with "looping" problem (it is possible that algorithm will fall into looping transferring same objects forward and backward in the same sets that may paralyze process) it is advised to use randomization – on each algorithm step, generally, there would be more than one object that should be transferred into some other group and idea here is to choose objects for "transferring" from all such objects randomly. To improve this technique, it is advised to scale the randomization probability function by some kind of error function – this way object that stronger violate Kuhn-Tucker conditions will be chosen with higher probability then object that lesser violate Kuhn-Tucker conditions.

Iterations amount highly depend on initial assumption quality. Simple initialization technique is a short iteration process: at first, some arbitrary point is chosen; then initialization algorithm looks for the nearest to previously chosen point point from another class; after repeating this procedure very quickly

converges to some pair of boundary points. These pair of points is used to fill I_S set (assumed to be a support objects set).

Also, in final implementation algorithm would be locked to not exceed 10000 iterations - tests shown that in some cases even mentioned randomization trick the algorithm fall into "semi-endless" state.

The high efficiency of the algorithm results from two facts:

1. Optimization that is solving on step 4, depends only on matrices Q_{SS} and Q_{CS}. That's mean the scalar product $< x_i, x_j >$ is computed only for pairs of objects such as "support-support" and "support-intruder"
2. Set I_S is changing only by 1 element on each step. This may help to reduce complexity of the inverse matrix Q_{SS}^{-1} "recalculating" from $O(h^3)$ to $O(h^2)$ by using the result obtained on previous step.

This algorithm (INCAS) should be good for cases with small amount of support vectors $h = |I_S|$.

PEGASOS – stochastic gradient descent

Another efficient method is the stochastic modification of the simple gradient descent algorithm. The main idea of the algorithm is to moving in the direction opposite of the gradient of the objective functional. But instead of using all the dataset in once this stochastic implementation proposes to consume points from the training set one-by-one. Also, objective function $G(w, \xi_i)$ in this case is rewritten to one-point form $\tilde{G}$ as follow:

$$\tilde{G}(w, \xi_i) = \xi_i + \frac{\mu}{2} < w, w >$$

$$\tilde{G}(w, b, i) = \big(1 - M_i(w, b)\big)_+ + \frac{\mu}{2} < w, w >$$

$$\tilde{G}(w, b,\ \ x_i, y_i) = \{\begin{matrix} 1 - y_i(< w, x_i > + b) & if\ 1 > y_i(< w, x_i > + b) \\ 0 & if\ 1 > y_i(< w, x_i > + b) \end{matrix}\} + \frac{\mu}{2} < w, w >$$

Gradients are:

$$\frac{d\tilde{G}(w, b,\ \ x_i, y_i)}{db} = \{\begin{matrix} -y_i & if\ 1 > y_i(< w, x_i > + b) \\ 0 & if\ 1 > y_i(< w, x_i > + b) \end{matrix}\}$$

$$\frac{d\tilde{G}(w, b,\ \ x_i, y_i)}{dw_j} = \{\begin{matrix} -y_i w_j x_{i,j} & if\ 1 > y_i(< w, x_i > + b) \\ 0 & if\ 1 > y_i(< w, x_i > + b) \end{matrix}\} + \mu w_j$$

Finally, on each algorithm step values w, b is increased by corresponding gradient scaled by learning rate: η_t

$$w_{j,t+1} = w_{j,t} - \eta_t \frac{d\tilde{G}(w, b,\ \ x_i, y_i)}{dw_j}$$

$$b_{t+1} = b_t - \eta_t \frac{d\tilde{G}(w, b,\ x_i, y_i)}{db}$$

$$\eta_t = \frac{1}{\mu t},$$

where t is the algorithm step counter. This way, learning rate is initialized by $\frac{1}{\mu}$ value and decreasing over steps.

"PEGASOS" method **pseudocode**

Input:
    X - training set
    μ - training parameter
Output:
    w, b - SVM classifier parameters

1: Initialization
    $w = 0,\ b = 0,\ t = 0,\ T = 0,\ G_t = 0$
2: **repeat**
3: $T = T + 1$
4: **randomize** $\boldsymbol{X}$
5: **for each** $\{\boldsymbol{x_i}, \boldsymbol{y_i}\} \in \boldsymbol{X}$
6: $t = t + 1,\ \eta = \frac{1}{\mu t}$
7: Update w, b parameters

$$w_j = w_j - \eta_t \frac{d\tilde{G}(w, b,\ x_i, y_i)}{dw_j}, j = 1, \dots, length(w)$$

$$b_{t+1} = b_t - \eta_t \frac{d\tilde{G}(w, b,\ x_i, y_i)}{db}$$

8: **end for**
9: estimate G_{t+1} objective function value

$$G_{t+1} = \frac{1}{l}\sum_{i=1}^{l} \max\{0, 1 - y_i^T(w^T x_i)\} + \frac{\mu}{2} w^T w, \qquad \{\boldsymbol{x_i}, \boldsymbol{y_i}\} \in \boldsymbol{X}$$

10: **if** $T \geq 10000$ or $\frac{|G_{t+1} - G_t|}{G_t} \leq 10^{-6}$ **then break else** $G_t = G_{t+1}$
11: **while** true

There are several exit conditions: one for iterations amount and another for accuracy. Both of them included in the algorithm to ensure a balance between quality and time.

PEGASOS vs INCAS

To test and compare both mentioned here algorithms, MATLAB scripts **timing_AS.m, timing_NOVEL.m** and **timing.m** had been implemented (code is given in Appendix section –Appendix A).

Both algorithms has been used for the same problem – **series** of classification of "Disputed Federalist Papers" dataset that has been split to two sub-sets: **training set**(86 objects) and **tuning set**(20 objects), with 70 features in each object. Each algorithm has been locked to not exceed the same amount of iteration steps – where the steps threshold is being changed in interval $[10 \dots 1000]$ during whole series with increment of 1. Each classification has been done 10 times and results has been averaged. This measure is primarily due to the presence stochasticity in both algorithms implementations.

Results are shown below in form of corresponding figures.

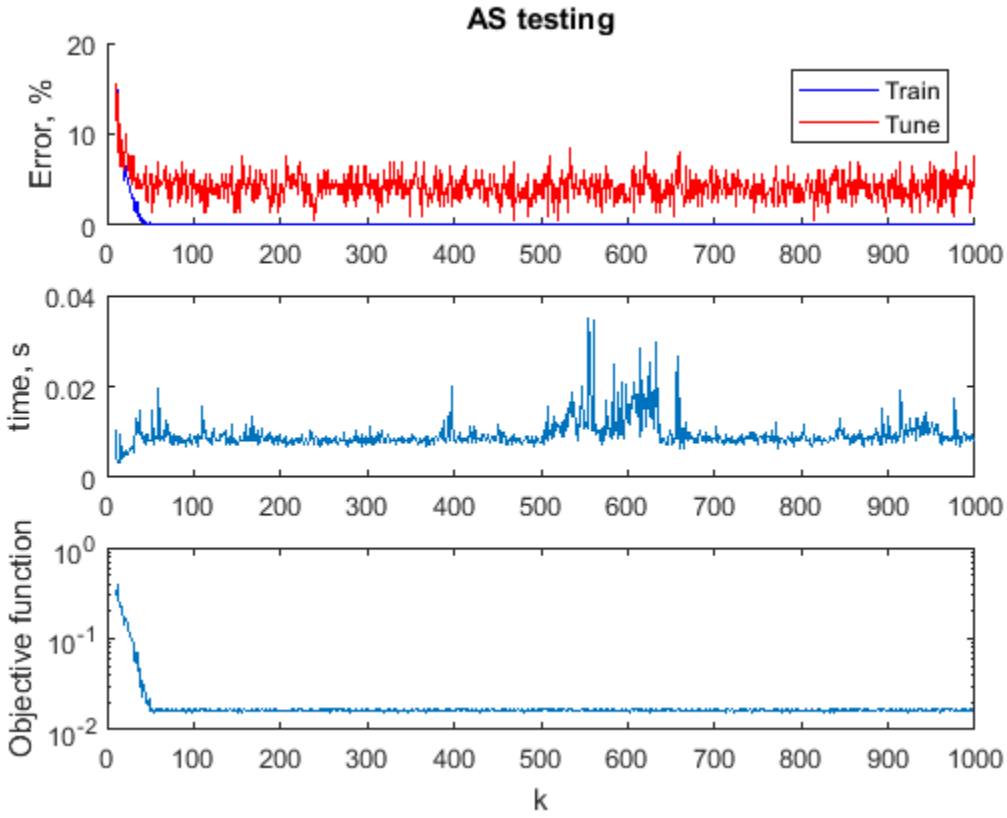

Figure 1 INCAS testing, $\mu = 0.1$

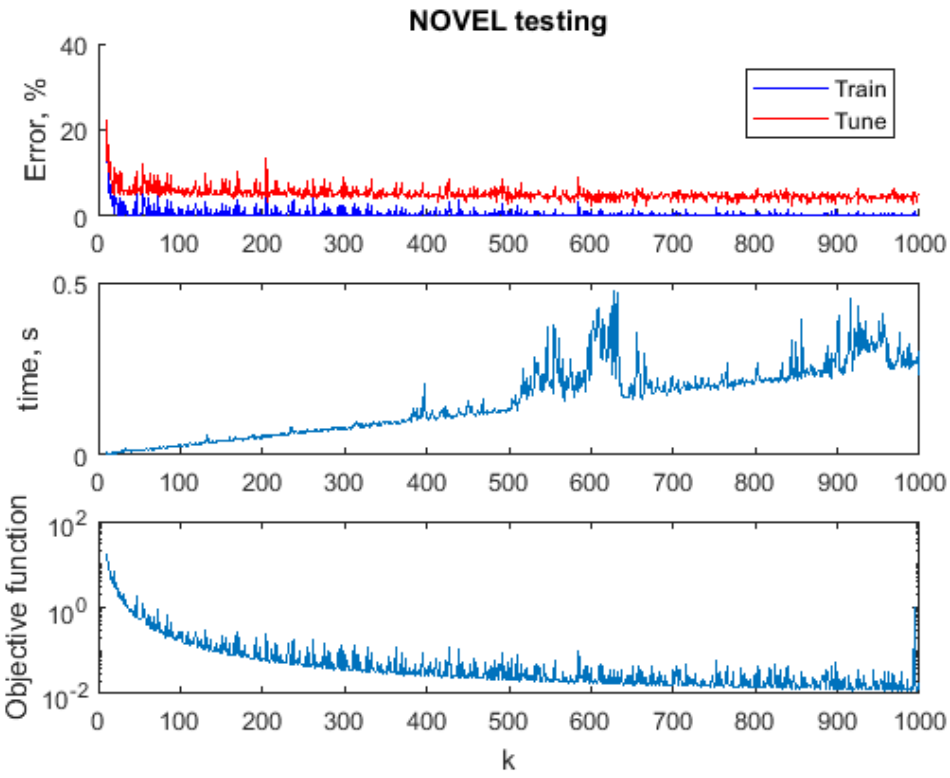

Figure 2 PEGASOS testing, $\mu = 0.1$

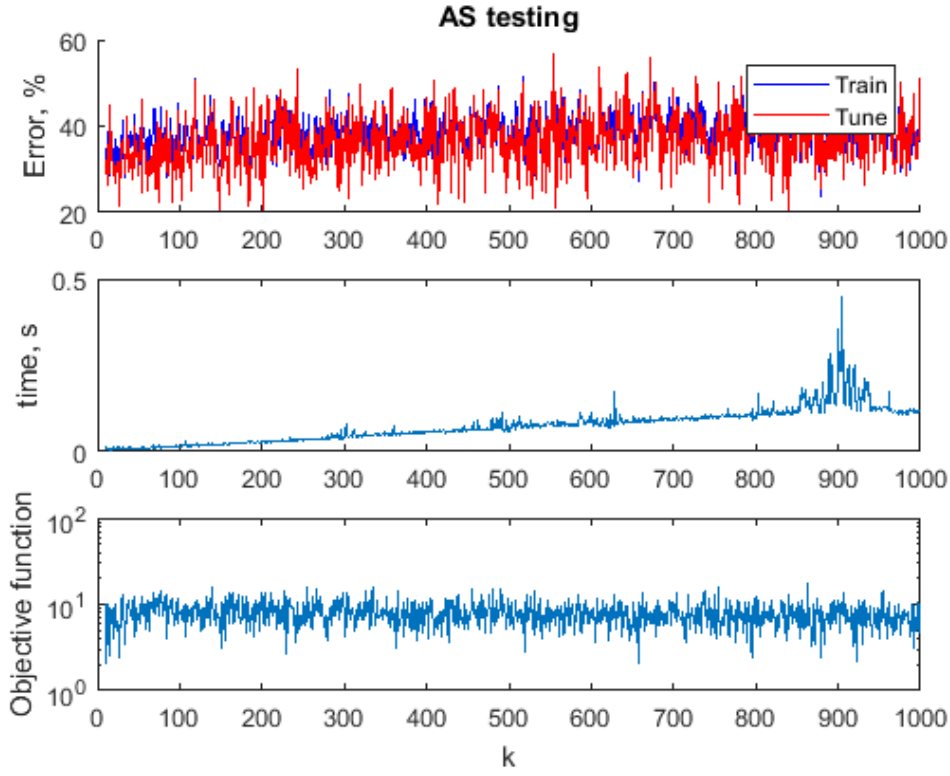

Figure 3 INCAS testing, $\mu = 1$

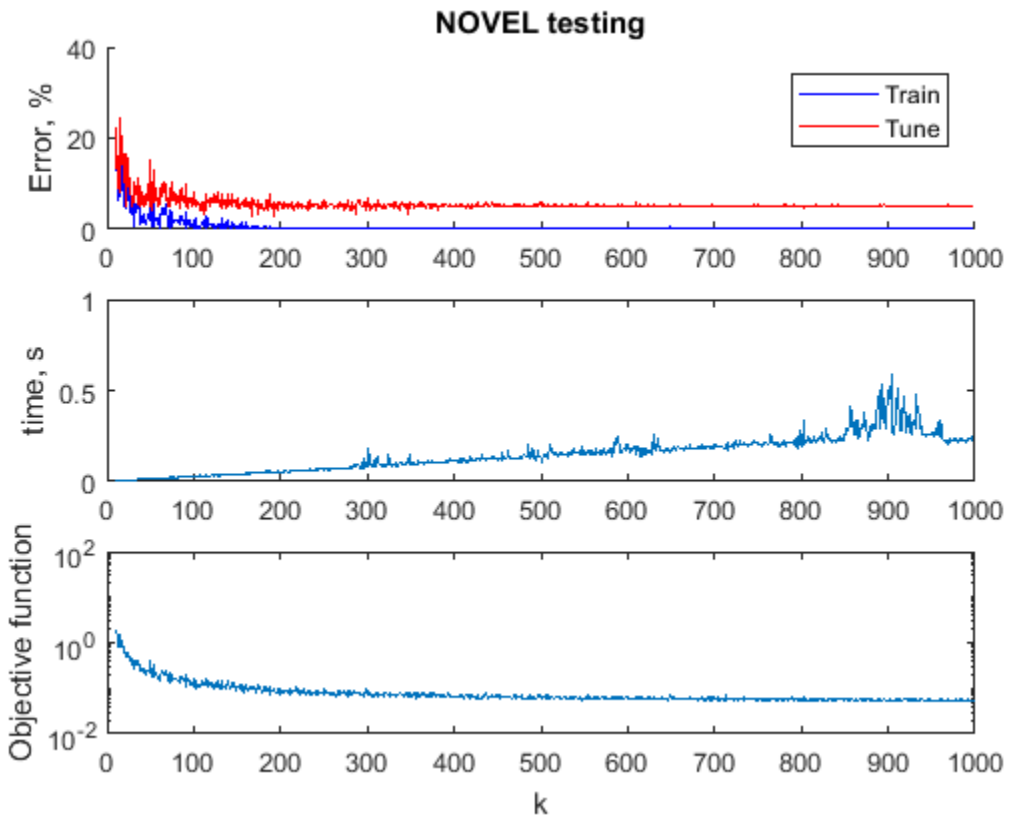

Figure 4 PEGASOS testing, $\mu = 1$

For smaller μ ($\mu = 0.1$) INCAS methods solves the problem faster than PEGASOS that is seen from fig.1 and fig.2. INCAS need only ~70 iterations to deal with it and that's why the execution time remains the same with k (iterations threshold) increasing. But for higher μ ($\mu = 1$) INCAS seems to become paralyzed and even 1000 steps is not enough to find a solution. Seems that INCAS may be considered as a solution for lower μ and PEGASOS for higher μ values.

Mentioned algorithms has been compared with built-in MATLAB solver "quadprog". Comparison has been implemented by **run_AS.m**, **run_NOVEL.m** and **run_quadprog.m** functions (that implement mentioned here **INCAS** and **PEGASOS** algorithms in form requested in the assignment) and **test_AS.m**, **test_NOVEL** and **test_quadprog.m** functions (that are wrappers for mentioned run_AS.m, run_NOVEL and run_quadprog.m). Code is attached in the Appendix section – Appendix B.

Result of algorithm comparison is given in tab.1.

Table 1 Solvers comparison

μ	Method	z	Train error, %	Tune error, %	Time, s
0.01	AS	0.001640	0.00	0.00	0.0573
	NOVEL	0.006830	0.00	0.00	2.4207
	quadprog	0.000447	0.00	5.00	0.2972
0.1	AS	0.015763	0.00	5.00	0.0262
	NOVEL	0.004875	0.00	5.00	2.6574
	quadprog	0.004330	0.00	5.00	0.0207
1	AS	2.499050	41.86	45.00	1.6278
	NOVEL	0.045566	0.00	5.00	2.3952
	quadprog	0.043149	0.00	5.00	0.0216

SVM application results

Q1:

Write MATLAB code to solve the QP to find the classifying hyperplane, where the matrices M and H are extracted from the training matrix train described above. Solve the QP for these values of μ: 0, 0.001, 0.01, 0.1, 1, 10, and 100. For each value of μ, calculate and print the following information:

- the optimal QP objective • the values of b and $||w||^2$
- the number of misclassified points (those that lie on the wrong side of the classifying hyperplane) from the training set, which is described in the matrix train
- the number of misclassified points from the tuning set, which is described in the matrix tune
- the predictions of authorship for the 12 disputed papers. (To calculate this pre diction, take each of the rows in test and determine which side of the classifying hyperplane they lie on.) Express your result with one line of output for each of the 12 papers, indicating the paper number, predicted author and margin (the value of $w^T x + b$ for the x corresponding to this paper.)

To answer this question, same **test_AS.m**, **test_NOVEL.m** and **test_quadprog.m** function were used.

Testing for $\mu = 0$ in case of **NOVEL (PEGASOS)** method had not been carried out – given realization of this method assumes $\mu > 0$ and doesn't work with $\mu = 0$.

Table 2 Q1 answer: AS results

μ	Z	b	$\|\|w\|\|_2$	p_1	p_2
0	0	-1.00	0.32	0	0
0.001	0.000144	-1.00	0.29	0	1
0.01	0.001686	-1.00	0.337	0	1
0.1	0.016795	-1.00	0.336	0	2
1	1.354126	0.66	0.153	23	6
10	3.256900	15.37	0.034	45	10
100	1.587153	12.28	0.011	44	9

Table 3 Q1 answer: NOVEL results

μ	Z	b	$\|\|w\|\|_2$	p_1	p_2
0.001	0.015678	-363.6	31.35	0	0
0.01	0.006664	-80.44	1.33	0	0
0.1	0.005638	3.84	0.11	0	1
1	0.045812	-0.88	0.09	0	1
10	0.235794	0.06	0.021	3	1
100	0.475247	0.01	0.0025	15	3

Table 4 Q1 answer: quadprog results

μ	Z	b	$\|\|w\|\|_2$	p_1	p_2
0	0	-4.79	0.086	0	1
0.001	0.000043	-4.79	0.086	0	1
0.01	0.000447	-4.79	0.086	0	1
0.1	0.004330	-4.79	0.086	0	1
1	0.043149	-4.79	0.086	0	1
10	0.229129	-2.62	0.022	3	1
100	0.473967	-0.42	0.0024	15	3

Table 5 Q1 answer: AS authorship prediction

$n\backslash\mu$	0	0.001	0.01	0.1	1	10	100
1	1	1	-1	1	1	-1	-1
2	-1	-1	-1	-1	1	-1	-1
3	-1	-1	-1	1	1	-1	1
4	-1	-1	-1	-1	1	-1	-1
5	-1	1	-1	-1	1	-1	1
6	-1	-1	-1	-1	1	-1	1
7	-1	-1	-1	1	1	-1	1
8	-1	-1	-1	-1	1	-1	-1
9	-1	-1	-1	-1	1	-1	-1
10	-1	-1	-1	1	1	-1	-1
11	-1	-1	-1	-1	1	-1	1
12	-1	-1	-1	-1	1	-1	1

Table 6 Q1 answer: NOVEL authorship prediction

$n\backslash\mu$	0.001	0.01	0.1	1	10	100
1	-1	-1	-1	-1	-1	1
2	-1	-1	-1	-1	-1	1
3	-1	-1	1	-1	1	1
4	-1	-1	-1	-1	-1	1
5	-1	-1	-1	-1	-1	1
6	-1	-1	-1	-1	1	1
7	-1	-1	-1	-1	-1	1
8	-1	-1	-1	-1	-1	1
9	-1	-1	-1	-1	-1	1
10	-1	-1	-1	-1	-1	1
11	-1	-1	-1	-1	1	1
12	-1	-1	-1	-1	-1	1

Table 7 Q1 answer: quadprog authorship prediction

$n\backslash\mu$	0	0.001	0.01	0.1	1	10	100
1	-1	-1	-1	-1	-1	-1	1
2	-1	-1	-1	-1	-1	-1	1
3	-1	-1	-1	-1	-1	1	1
4	-1	-1	-1	-1	-1	-1	1
5	-1	-1	-1	-1	-1	-1	1
6	-1	-1	-1	-1	-1	1	1
7	-1	-1	-1	-1	-1	-1	1
8	-1	-1	-1	-1	-1	-1	1
9	-1	-1	-1	-1	-1	-1	1
10	-1	-1	-1	-1	-1	-1	1
11	-1	-1	-1	-1	-1	1	1
12	-1	-1	-1	-1	-1	-1	1

Q2:

What is the effect of μ on $||w||^2$ and on the quality of the classifying hyperplane? What is the best value of μ?

Increasing of μ leads to decreasing of $||w||_2$ and accuracy (misclassification increases). It is quite expected, because objective functional that includes both $||w||_2$ and errors becomes more dependent from $||w||_2$ with μ increasing and so balance between errors(misclassifications) and "SVM margin" (distance between "classes" hyperplanes) is shifted to the second one. Seems that $\mu = 0.1$ or $\mu = 1$ is good enough, because it provides same errors as in case smaller values of μ but in the same time it provides greater "SVM margin" $||w||_2$.

Q3:

Fix $\mu = 0.1$ for purposes of this question and the next. Suppose you want to use only 2 of the 70 attributes (word frequencies) for your prediction of authorship. Determine which pair of attributes is the most effective in determining a correct prediction as follows. For each of the 2415 pairs of possible attributes, determine a separating hyperplane in R^2 . (Note that for these problems, $n = 2$ in our formulation above in contrast to $n = 70$ when we use all features.) For each plane, determine the number of misclassified cases from the tuning set. Report the best classified, and the two words that it uses, along with the number of misclassified tuning points for this classifier.

To answer this question, **Q3.m** MATLAB script has been implemented (code is attached in the appendix section – appendix C). This script goes by all features pair and estimate tuning misclassification in sense of amount of misclassified points of tuning set. Results of it execution is given below in form of figures each point of which is equal to amount of misclassified tuning points while using pair of corresponding features. By red circle the optimal features pair is marked. It is clearly seen that presence of 60's feature in feature's pairs almost guaranties good "separation" – figures below has back line against index with value 60. 60 corresponds to word "upon". Same as in [5], "upon" feature provides the best separation. The word "a" has been chosen as second feature. Pair of (60,1) or "upon", "a" just were the first pair of

features with smallest tuning misclassification rate but they were not unique some other pairs of “upon”,”*” also had the same tuning error (zero misclassified points) as it evident from fig. 5.

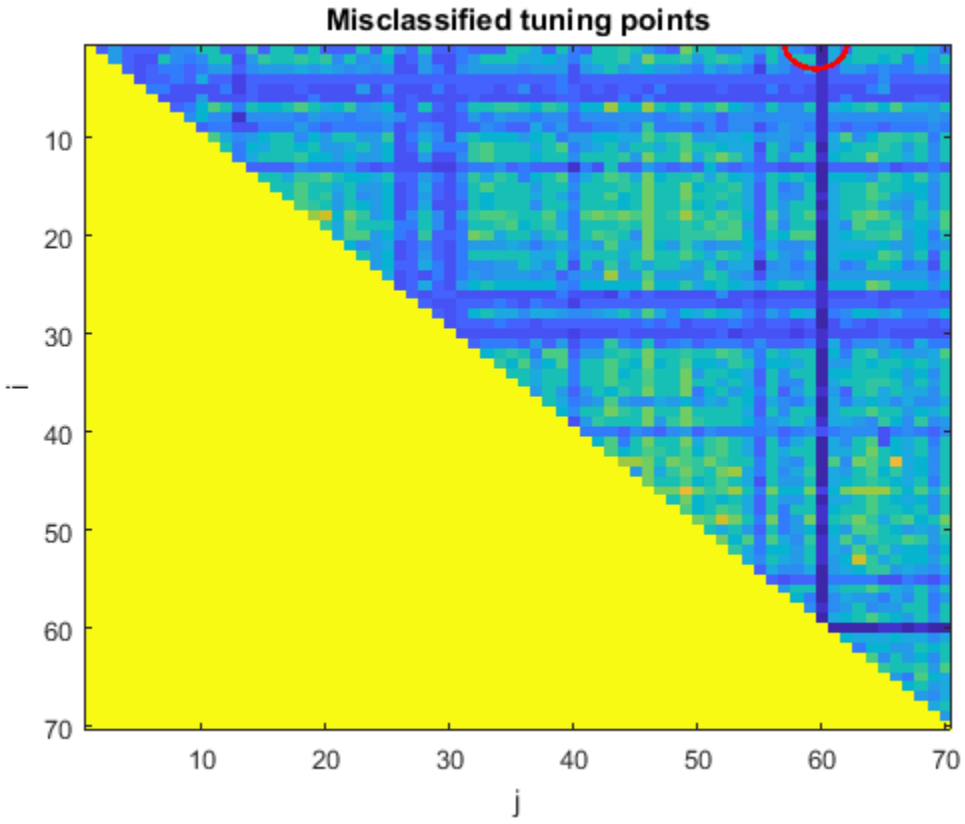

Figure 5 Optimal pair of features as for NOVEL method: i= 1, j= 60, p2= 0

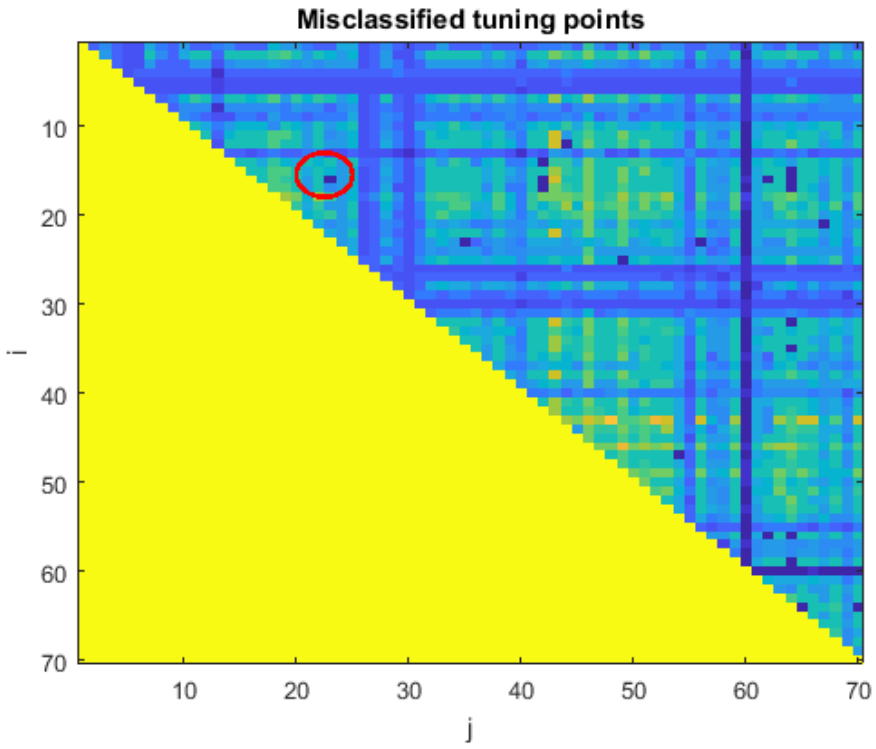

Figure 6 Optimal pair of features as for quadprogram method: i= 16, j= 23, p2= 0

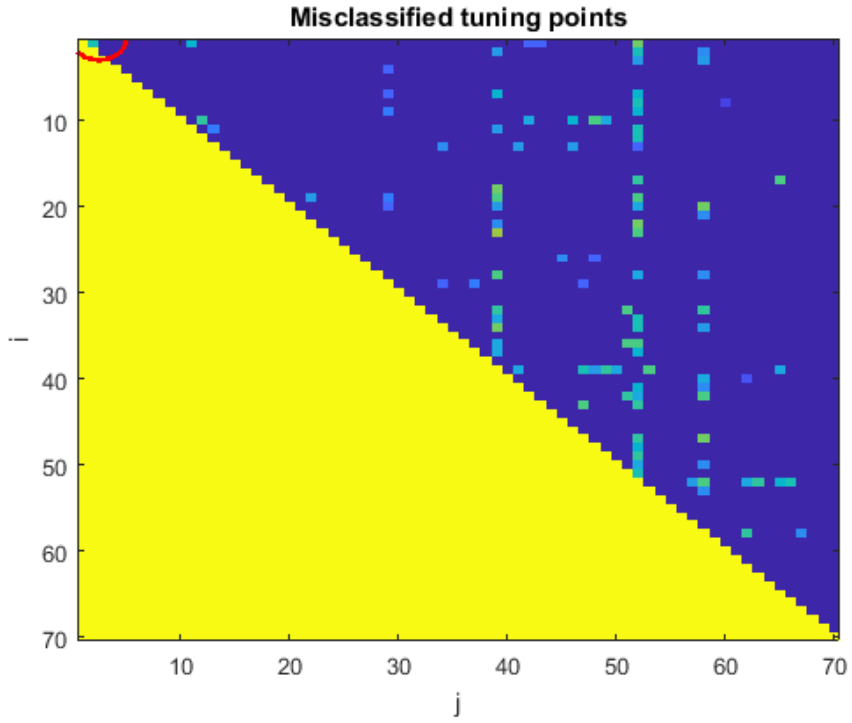

Figure 7 Optimal pair of features as for AS method i= 1, j= 3, p2= 0

Q4:

Use the optimal classifier from $Q3$ to predict the authorship of each of the twelve papers. Plot all the data points according to the "best" two attributes on a two dimensional figure using MATLAB built-in plotting routines. Use 'o' for Hamilton papers and '+' for Madison papers, and '*' for disputed papers in the plot. Use MATLAB to draw in the calculated line $w^T x = -b$. Check to see if the number of misclassified papers shown in your plot agrees with your calculations. (Note that some points may coalesce, so you may want to randomly perturb the points by a small amount to visualize all these points).

To answer this question, **Q4.m** MATLAB script has been implemented (the code is attached in Appendix section – Appendix C) using results of NOVEL algorithm obtained in Q3:

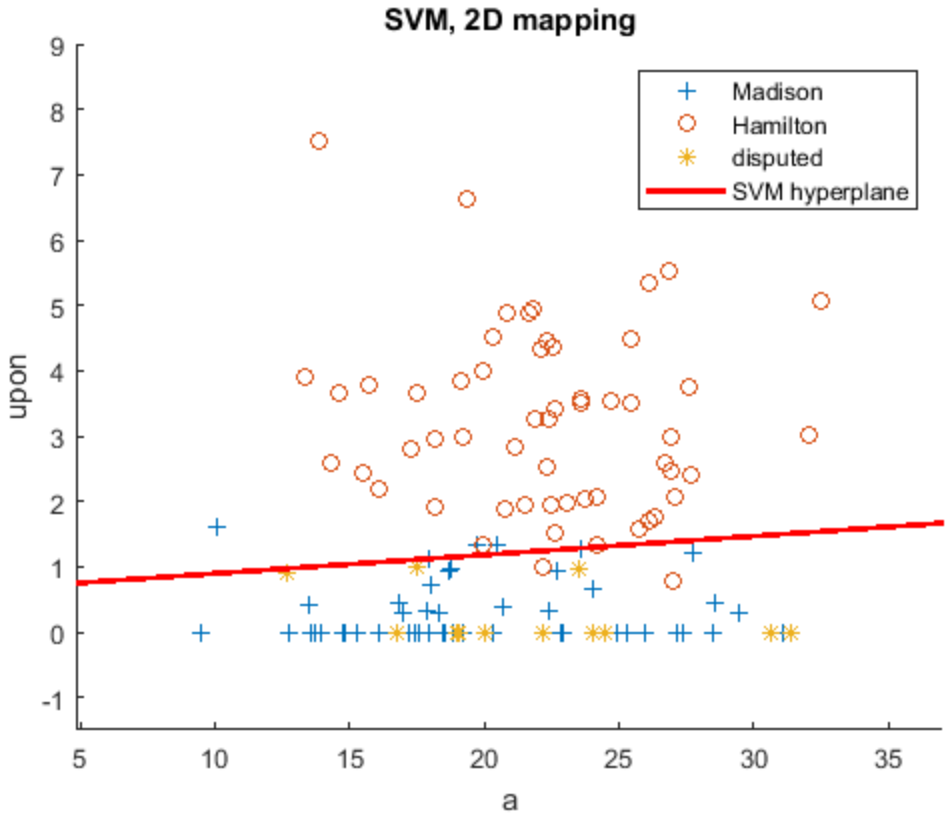

Figure 8 Optimal pair of features as for NOVEL method: i= 1, j= 60 ("a","upon")

It is seen that all "disputed" objects lie in the "Madison" area here on the fig.8.

Conclusions

Incremental active set method which algorithm is given by pseudo code on in appropriate "Algorithms" section of this paper had stacked or paralyzed on the federalist dataset. On the step where it should transfer objects that violating Kuhn-Tacker conditions to an appropriate group, it had been going by circles transferring the same object from "support vector" (I_S) to "intruders/peripheral"(I_c, I_O) and then returning it to "support vector". This problem was solved by incorporating the stochasticity – in the used realization of the given algorithm instead of taking the first object that violates mentioned conditions the program has taken random object where probability that given object would be chosen for transferring on the current step was scaled by how strong the Kuhn-Tacker condition violated by the given object. Example of the probability function built for I_O to I_S transferring is show below on fig.9. (Object from I_O with $M \leq 1$ is transferred to I_S):

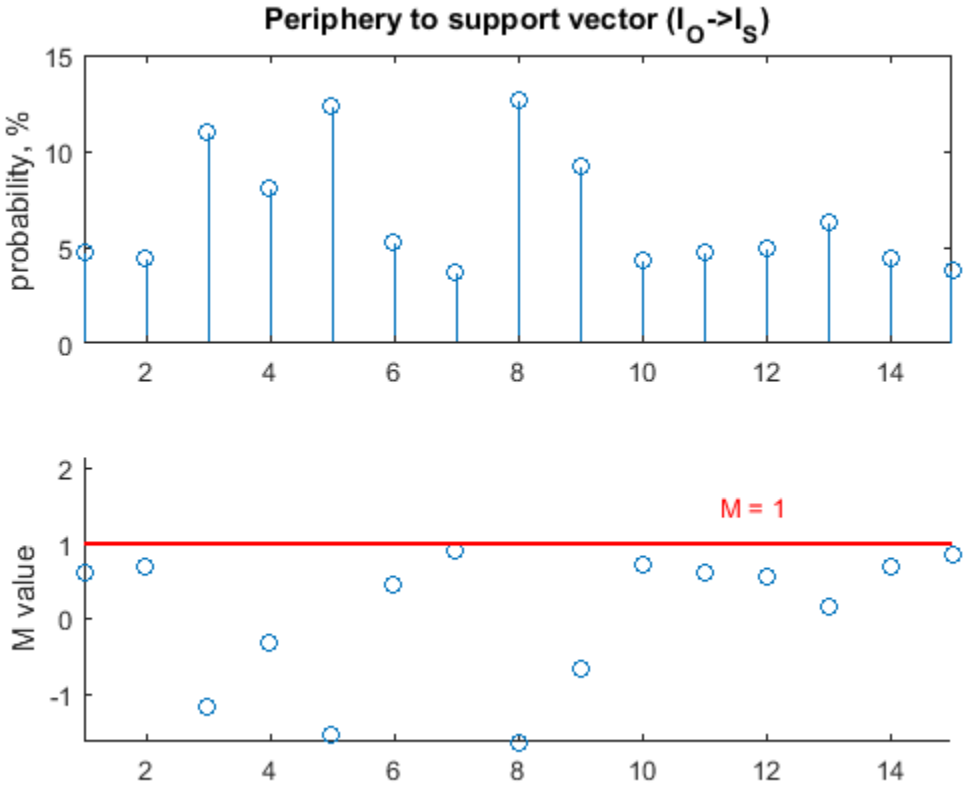

Figure 9 O to S transfer randomization

After modification described above, INCAS method has shown itself like pretty fast SVM training technique – for smaller $\mu < 1$ it worked good and fast. By the way, it may work even faster if matrix Q_{SS} inversion would be done using the results of it's inversion on previous step: it is possible due to the fact that I_S is changing only by 1 element per iteration and so Q_{SS} is changing by 1 column and 1 row. But for greater values of $\mu \geq 1$ INCAS refused to converge by 10 000 iterations.

Gradient descent algorithm has shown nice converging rate, but for smaller $\mu < 1$ INCAS is better in sense of both : time and error.

Results of SVM application to federalist dataset using mentioned here training techniques correlates with results obtained in [5]. “upon” word frequency is the most informative feature for authorship classification that is shown by fig.5, fig.6 and fig.8.

Appendix A

timing_AS.m

```
function [b,w,time] = timing_AS(x,y,mu,k)

    tic
    %% Preparations
        % Prepare Q matrix
    Q = (y*y').*(x'*x);

        % Initial assumption
            % Support vectors
    I_s = zeros(size(y,1),1);
            % Periphery vectors
    I_o = ones(size(y,1),1);
            % Violator vector
    I_c = zeros(size(y,1),1);

            % Find initial support vectors
    i1 = randi([1 length(I_s)]);
    i1_previous = 0;
    while 1
        y_n_ind = find(y~=y(i1));
        r = sum((x(:,y_n_ind)-x(:,i1)).^2);
        i2 = y_n_ind(find(r==min(r),1));
        if(i1_previous == i2)
            break;
        end
        i1_previous = i1;
        i1 = i2;
    end

    I_s([i1 i2]) = 1;
    I_o([i1 i2]) = 0;

        % Define C
    C = 1 / (mu * size(y,1));

        % Variable for iterations count
    iter_count = 0;

    %% Solving
    flag2 = 1;

    while flag2
        flag1 = 1;
        while flag1
                % Construct S/C matrices
            e_s = ones(sum(I_s),1);
            e_c = ones(sum(I_c),1);
            Q_cs = Q(I_c==1, I_s==1);
            Q_ss = Q(I_s==1, I_s==1);

                % lamda optimization
```

```
            lamda_s = 2 * (e_s' - C*(e_c')*Q_cs) * (Q_ss^(-1));

            flag1 = 0;

            i_s = find(I_s==1);
                % Decompose vectors
            if(sum(I_s)>2)
                    % From support vector to periphery
                i_s_0 = i_s(lamda_s <= 0);
                if(size(i_s_0,1)>0)
                    lamda_s_0 = lamda_s(lamda_s<=0);
                    p_s_0 = 0.5*(ones(size(lamda_s_0))/(length(lamda_s_0)))
+...
                            0.5 * ((0-lamda_s_0)/sum(0-lamda_s_0));
                    i_s_0_chosen = i_s_0(1+sum(rand >= cumsum(p_s_0)));

                    I_s(i_s_0_chosen) = 0;
                    I_o(i_s_0_chosen) = 1;

                    flag1 = 1;
                    iter_count = iter_count + 1;
                        % If some decompositions happened
                        % continue the cycle
                    continue;
                end

                    % From support vector to violator
                i_s_C = i_s(lamda_s >= C);

                if(size(i_s_C,1)>0)
                    lamda_s_C = lamda_s(lamda_s>=C);
                    p_s_C = 0.5*(ones(size(lamda_s_C))/(length(lamda_s_C)))
+...
                            0.5*((lamda_s_C-C) / sum(lamda_s_C-C));

                    i_s_C_chosen = i_s_C(1+sum(rand >= cumsum(p_s_C)));
                    I_s(i_s_C_chosen) = 0;
                    I_c(i_s_C_chosen) = 1;

                    flag1 = 1;
                    iter_count = iter_count + 1;
                    continue;
                end
            end

        end
            % Estimate SVM coefficients
        w = x(:,I_s==1) * ((lamda_s').*y(I_s==1));
        b = median(y(I_s==1)' - (w')*x(:,I_s==1));

            % Break the optimization on "k" step
        if(iter_count>=k)
            break;
        end
```

```
        flag2 = 0;

        i_o = find(I_o==1);
        i_c = find(I_c==1);
            % Estimate indent
        M_o = (y(i_o)') .* ((w')*x(:,i_o) + b);
        M_c = (y(i_c)') .* ((w')*x(:,i_c) + b);

            % Decompose vectors

            % From periphery to support vector
        i_o_less = i_o(M_o <= 1);

        if(size(i_o_less,1)>0)
            M_o_less = M_o(M_o<=1);
            p_o_less = 0.5*(ones(size(M_o_less))/(length(M_o_less))) +...
                    0.5 * ((1- M_o_less)/sum(1- M_o_less));
            i_o_less_chosen = i_o_less(1+sum(rand >= cumsum(p_o_less)));

            I_o(i_o_less_chosen) = 0;
            I_s(i_o_less_chosen) = 1;

            flag2 = 1;
            iter_count = iter_count + 1;
                % If some decompositions happened
                % continue the cycle
            continue;
        end

            % From intruder to support vector
        i_c_more = i_c(M_c >= 1);

        if(size(i_c_more,1)>0)
            M_c_more = M_c(M_c>=1);
            p_c_more = 0.5*(ones(size(M_c_more))/(length(M_c_more))) +...
                      0.5 * ((M_c_more-1)/sum(M_c_more-1));
            i_c_more_chosen = i_c_more(1+sum(rand >= cumsum(p_c_more)));

            I_c(i_c_more_chosen) = 0;
            I_s(i_c_more_chosen) = 1;

            flag2 = 1;
            iter_count = iter_count + 1;
            continue;
        end


    end

 %% Finalization
     % Estimate execution time
    time = toc;
end
```

timing_NOVEL.m

```
function [b,w,time] = timing_NOVEL(x,y,mu,k)

    tic

 %% Preparations

        % Define the objective function
    Goal = @(y,x,mu,w,b)(1/length(y)) * sum( max([zeros(1,length(y));1-
(y').*(w'*x+b)]) ) + (mu/2)*(w')*w;

        % Initial assumption
    w = zeros(size(x,1),1);
    b = 0;

        % Initialize solution/solver
    t = 0;

%% Solving
Goal_before = 0;
for iter_count = 1:10000
    I = randperm(length(y));

    for j=1:length(y)
        t = t+1;
        etta = 1.0/(mu*t);

        error_flag = (1-(y(I(j)))*(w'*x(:,I(j))+b))>0;

        % Gradient step for all "w" elements
        for i = 1:size(w,1)
            if(error_flag)
                w(i) = w(i) - etta * (-y(I(j))*x(i,I(j)) + mu*w(i));
            else
                w(i) = w(i) - etta * ( mu*w(i));
            end
        end

        % Gradient step for "b"
        if(error_flag)
            b = b - etta*(-y(I(j)));
        end

    end

    Goal_then = Goal(y,x,mu,w,b);
    if(abs((Goal_before - Goal_then))/Goal_before <10^(-6))
        break;
    end
    Goal_before = Goal_then;

    if(iter_count>=k)
        break;
```

```
    end
end


 %% Finalization
    % Estimate execution time
    time = toc;
end
```

timing.m

```
clear all;
close all;
clc;

%% Preparations
% Load the data
[train,tune,test,dataDim] = getFederalistData;

        % Parse the data
    y = [train(:,1); tune(:,1)];
    y(y==2)=-1;
    x = [train(:,2:end); tune(:,2:end)]';

    M = x(:,y==-1); % M is the set of objects of 1 class (Madison)
    H = x(:,y==1);  % H is the set of objects of 2 class (Hamilton)

    x = [M(:,11:end) H(:,11:end)];
    y = [-ones(size(M(:,11:end),2),1); ones(size(H(:,11:end),2),1)];
            % tuning set (for tuning set - 10 points from M and 10 from H)
    x_tune = [M(:,1:10) H(:,1:10)];
    y_tune = [-ones(size(M(:,1:10),2),1); ones(size(H(:,1:10),2),1)];

%% Mine the statistics data
k = 10:1000;
mu = 0.1;
time_AS = zeros(size(k));
error_train_AS = zeros(size(k));
error_tune_AS = zeros(size(k));
z_AS = zeros(size(k));

time_NOVEL = zeros(size(k));
error_train_NOVEL = zeros(size(k));
error_tune_NOVEL = zeros(size(k));
z_NOVEL = zeros(size(k));

for i=1:length(k)
    for j=1:10
        % Test AS with given k-steps constraint
        [b,w,time_k] = timing_AS(x,y,mu,k(i));
        time_AS(i) = time_AS(i)+ time_k;
        error_train_AS(i) = error_train_AS(i) + 100*(1-
sum(y==(((w'*x+b)>=0)*2-1)')/size(y,1));
```

```
        error_tune_AS(i) = error_tune_AS(i) + 100*(1-
sum(y_tune==(((w'*x_tune+b)>=0)*2-1)')/size(y_tune,1));
        M = (y').*((w')*x+b);
        E = max([(1 - M); zeros(size(M))]);
        z_AS(i) = z_AS(i) + (1/length(y))*sum(E) + (mu/2)*(w'*w);
    end
    time_AS(i) = time_AS(i)/10;
    error_train_AS(i) = error_train_AS(i)/10;
    error_tune_AS(i) = error_tune_AS(i)/10;
    z_AS(i) = z_AS(i)/10;

    for j=1:10
        % Test AS with given k-steps constraint
        [b,w,time_k] = timing_NOVEL(x,y,mu,k(i));
        time_NOVEL(i) = time_NOVEL(i)+ time_k;
        error_train_NOVEL(i) = error_train_NOVEL(i) + 100*(1-
sum(y==(((w'*x+b)>=0)*2-1)')/size(y,1));
        error_tune_NOVEL(i) = error_tune_NOVEL(i) + 100*(1-
sum(y_tune==(((w'*x_tune+b)>=0)*2-1)')/size(y_tune,1));
        M = (y').*((w')*x+b);
        E = max([(1 - M); zeros(size(M))]);
        z_NOVEL(i) = z_NOVEL(i) + (1/length(y))*sum(E) + (mu/2)*(w'*w);
    end
    time_NOVEL(i) = time_NOVEL(i)/10;
    error_train_NOVEL(i) = error_train_NOVEL(i)/10;
    error_tune_NOVEL(i) = error_tune_NOVEL(i)/10;
    z_NOVEL(i) = z_NOVEL(i)/10;
end

%% Present mined results (plot figures)

    % Plot AS figures
figure;

subplot(3,1,1);
hold on;
title('AS testing');
plot(k,error_train_AS,'b');
plot(k,error_tune_AS,'r');
ylabel('Error, %');
legend({'Train','Tune'});


subplot(3,1,2);
plot(k,time_AS);
ylabel('time, s');

subplot(3,1,3);
semilogy(k,z_AS);
xlabel('k');
ylabel('Objective function');


    % Plot NOVEL figures
figure;
```

```
subplot(3,1,1);
hold on;
title('NOVEL testing');
plot(k,error_train_NOVEL,'b');
plot(k,error_tune_NOVEL,'r');
ylabel('Error, %');
legend({'Train','Tune'});


subplot(3,1,2);
plot(k,time_NOVEL);
ylabel('time, s');

subplot(3,1,3);
semilogy(k,z_NOVEL);
xlabel('k');
ylabel('Objective function');
```

Appendix B

run_AS.m

```
function [z,b,w,p1,p2] = run_AS(M,H,mu)
    %% Preparations
        % Parse the data
            % training set
    x = [M(:,11:end) H(:,11:end)];
    y = [-ones(size(M(:,11:end),2),1); ones(size(H(:,11:end),2),1)];
            % tuning set (for tuning set - 10 points from M and 10 from H)
    x_tune = [M(:,1:10) H(:,1:10)];
    y_tune = [-ones(size(M(:,1:10),2),1); ones(size(H(:,1:10),2),1)];

        % Prepare Q matrix
    Q = (y*y').*(x'*x);

        % Initial assumption
            % Support vectors
    I_s = zeros(size(y,1),1);
            % Periphery vectors
    I_o = ones(size(y,1),1);
            % Violator vector
    I_c = zeros(size(y,1),1);

            % Find initial support vectors
                % 1)Take some random object and store it in i1
    i1 = randi([1 length(I_s)]);
    i1_previous = 0;
                % (not more then 100 steps for this finding procedure,
                % usually it is converged by few iterations)
    for l=1:100
                % 2)Estimate the distance between all other-class-objects to
                % to the i1 object
        y_n_ind = find(y~=y(i1));
        r = sum((x(:,y_n_ind)-x(:,i1)).^2);
                % 3)Choose as optimal pair for i1 such i2 object that
                % is located on minimal distance from i1.
        i2 = y_n_ind(find(r==min(r),1));
                % 4)If convergence reached - break the process...
        if(i1_previous == i2)
            break;
        end
                % ... if convergence not reached yet - continue the process
                % for i2 object.
        i1_previous = i1;
        i1 = i2;
    end

            % Use found i1 and i2 pair of objects from different classes
            % as initialization for S set.
    I_s([i1 i2]) = 1;
    I_o([i1 i2]) = 0;

        % Define C
    C = 1 / (mu * size(y,1));
```

```
        % Variable for iterations count
    iter_count = 0;

    %% Solving
    flag2 = 1;

    while flag2
        % In case of "endless" cycling - break it
        if(iter_count>=10000)
            break;
        end

        flag1 = 1;
        %fprintf(1,'=========\n');
        while flag1
                % Construct S/C matrices
            e_s = ones(sum(I_s),1);
            e_c = ones(sum(I_c),1);
            Q_cs = Q(I_c==1, I_s==1);
            Q_ss = Q(I_s==1, I_s==1);

                % lamda optimization
            lamda_s = 2 * (e_s' - C*(e_c')*Q_cs) * (Q_ss^(-1));

            flag1 = 0;

            i_s = find(I_s==1);
                % Decompose vectors
            if(sum(I_s)>2)
                    % From support vector to periphery
                i_s_0 = i_s(lamda_s <= 0);
                if(size(i_s_0,1)>0)
                    % Chose object for decomposition (between S and O sets)
                    % randomly but with modulated probability p_s_0.
                    % The probability function p_s_0 is chosen in such way
                    % that objects that strongly violates "lamda_s<0"
                    % condition will be chosen with higher probability
                    % (it has higher priority).
                    lamda_s_0 = lamda_s(lamda_s<=0);
                    p_s_0 = 0.5*(ones(size(lamda_s_0))/(length(lamda_s_0)))
+...
                            0.5 * ((0-lamda_s_0)/sum(0-lamda_s_0));
                    i_s_0_chosen = i_s_0(1+sum(rand >= cumsum(p_s_0)));

                    I_s(i_s_0_chosen) = 0;
                    I_o(i_s_0_chosen) = 1;
        %            fprintf(1,'S->O: %d \n',i_s_0_chosen);
                    flag1 = 1;
                    iter_count = iter_count + 1;
                        % If some decompositions happened
                        % continue the cycle
                    continue;
                end
```

```
                    % From support vector to violator
                i_s_C = i_s(lamda_s >= C);

                if(size(i_s_C,1)>0)
                    lamda_s_C = lamda_s(lamda_s>=C);
                    p_s_C = 0.5*(ones(size(lamda_s_C))/(length(lamda_s_C)))
+...
                            0.5*((lamda_s_C-C) / sum(lamda_s_C-C));

                    i_s_C_chosen = i_s_C(1+sum(rand >= cumsum(p_s_C)));
                    I_s(i_s_C_chosen) = 0;
                    I_c(i_s_C_chosen) = 1;
         %           fprintf(1,'S->C: %d \n',i_s_C_chosen);
                    flag1 = 1;
                    iter_count = iter_count + 1;
                    continue;
                end
            end

        end
    %   fprintf(1,'++++++++\n');
            % Estimate SVM coefficients
        w = x(:,I_s==1) * ((lamda_s').*y(I_s==1));
        b = median(y(I_s==1)' - (w')*x(:,I_s==1));

        flag2 = 0;

        i_o = find(I_o==1);
        i_c = find(I_c==1);
            % Estimate indent
        M_o = (y(i_o)') .* ((w')*x(:,i_o) + b);
        M_c = (y(i_c)') .* ((w')*x(:,i_c) + b);

            % Decompose vectors

            % From periphery to support vector
        i_o_less = i_o(M_o <= 1);

        if(size(i_o_less,1)>0)
            M_o_less = M_o(M_o<=1);
            p_o_less = 0.5*(ones(size(M_o_less))/(length(M_o_less))) +...
                    0.5 * ((1- M_o_less)/sum(1- M_o_less));
            i_o_less_chosen = i_o_less(1+sum(rand >= cumsum(p_o_less)));

            I_o(i_o_less_chosen) = 0;
            I_s(i_o_less_chosen) = 1;
        %    fprintf(1,'O->S: %d \n',i_o_less_chosen);
            flag2 = 1;
            iter_count = iter_count + 1;
                % If some decompositions happened
                % continue the cycle
            continue;
        end

            % From violator to support vector
```

```
        i_c_more = i_c(M_c >= 1);

        if(size(i_c_more,1)>0)
            M_c_more = M_c(M_c>=1);
            p_c_more = 0.5*(ones(size(M_c_more))/(length(M_c_more))) +...
                       0.5 * ((M_c_more-1)/sum(M_c_more-1));
            i_c_more_chosen = i_c_more(1+sum(rand >= cumsum(p_c_more)));

            I_c(i_c_more_chosen) = 0;
            I_s(i_c_more_chosen) = 1;
      %      fprintf(1,'C->S: %d \n',i_c_more_chosen);
            flag2 = 1;
            iter_count = iter_count + 1;
            continue;
        end


    end

    % Estimate requested z, p1, p2.
        % At first, estimate indents
    M_full = (y').*((w')*x+b);
        % Then errors
    E = max([(1 - M_full); zeros(size(y'))]);
        % Finally, z - objective function
    z = (1/length(y))*sum(E) + (mu/2)*(w'*w);
        % misclassified points - points with negative indent
        % (here comparison with "machine zero" is done instead of
        % real zero)
    p1 = sum(M_full < 1e-10);
        % estimate misclassified points for tuning set -
        % points with negative indent
        % (here comparison with "machine zero" is done instead of
        % real zero)
    M_full = (y_tune').*((w')*x_tune+b);
    p2 =  sum(M_full < 1e-10);
end
```

run_Novel.m

```
function [z,b,w,p1,p2] = run_NOVEL(M,H,mu)
    %% Preparations
        % Parse the data
            % training set
    x = [M(:,11:end) H(:,11:end)];
    y = [-ones(size(M(:,11:end),2),1); ones(size(H(:,11:end),2),1)];
            % tuning set (for tuning set - 10 points from M and 10 from H)
    x_tune = [M(:,1:10) H(:,1:10)];
    y_tune = [-ones(size(M(:,1:10),2),1); ones(size(H(:,1:10),2),1)];

        % Initial assumption
    w = zeros(size(x,1),1);
    b = 0;
```

```
        % Initialize solution/solver
    t = 0;

        % Define the objective function
    Goal = @(y,x,mu,w,b)(1/length(y)) * sum( max([zeros(1,length(y));1-
(y').*(w'*x+b)]) ) + (mu/2)*(w')*w;

%% Solving
Goal_before = 0;
for iter_count = 1:10000
    % Each step, the algorithm will run stochastic gradient descent
    % stepping all objects in random order. Here the order given by I.
    I = randperm(length(y));

    for j=1:length(y)
        t = t+1;
        % Learning rate decreasing with each stochastic step
        % (with initial value of 1/mu).
        etta = 1.0/(mu*t);

        % Error flag defines which sub-gradient would be used
        error_flag = (1-(y(I(j)))*(w'*x(:,I(j))+b))>0;

        % Gradient step for all "w" elements
        for i = 1:size(w,1)
            % On respect to error_flag one or another sub-gradient would be
            % chosen.
            if(error_flag)
                w(i) = w(i) - etta * (-y(I(j))*x(i,I(j)) + mu*w(i));
            else
                w(i) = w(i) - etta * ( mu*w(i));
            end
        end

        % Gradient step for "b"
        if(error_flag)
            b = b - etta*(-y(I(j)));
        end
    end

    Goal_then = Goal(y,x,mu,w,b);
    if(abs((Goal_before - Goal_then))/Goal_before <10^(-6))
        break;
    end
    Goal_before = Goal_then;

end
 %% Finalization
    % Estimate requested z, p1, p2.
        % At first, estimate indents
    M_full = (y').*((w')*x+b);
        % Then errors
    E = max([(1 - M_full); zeros(size(y'))]);
        % Finally, z - objective function
    z = (1/length(y))*sum(E) + (mu/2)*(w'*w);
```

```
        % misclassified points - points with negative indent
        % (here comparison with "machine zero" is done instead of
        % real zero)
    p1 = sum(M_full < 1e-10);
        % estimate misclassified points for tuning set -
        % points with negative indent
        % (here comparison with "machine zero" is done instead of
        % real zero)
    M_full = (y_tune').*((w')*x_tune+b);
    p2 =  sum(M_full < 1e-10);
end
```

run_quadprog.m

```
function [z,b,w,p1,p2] = run_quadprog(M,H,mu)
    % Parse the data
        % training set
    x = [M(:,11:end) H(:,11:end)];
    y = [-ones(size(M(:,11:end),2),1); ones(size(H(:,11:end),2),1)];
            % tuning set (for tuning set - 10 points from M and 10 from H)
    x_tune = [M(:,1:10) H(:,1:10)];
    y_tune = [-ones(size(M(:,1:10),2),1); ones(size(H(:,1:10),2),1)];

            % Estimate C
    C = 1 / (mu * size(y,1));

        % Parse 'SVM' data into QP problem
    Q = x'*x;
    Y = diag(y);
    H = Y*Q*Y + 1e-6*eye(length(y));
    f = -ones(size(y));
    a = y';
    K = 0;
    Kl = zeros(size(y));
    Ku = C * ones(size(y));

        % Solve by quadprog
    options = optimset('Display','off');
    lamda = quadprog(H,f,[],[],a,K,Kl,Ku,[],options);

    w = x * (lamda.*y);

    e = 1e-6;
    ind = find(lamda > e & lamda < C-e);
    b = mean(y(ind) - x(:,ind)'*w);

            % To estimate objective function, at first
            % estimate indent M
    M_full = (y').*((w')*x+b);
            % Then errors
    E = max([(1 - M_full); zeros(size(y'))]);
            % Finally, z - objective function
    z = (1/length(y))*sum(E) + (mu/2)*(w'*w);
            % misclassified points - points with negative indent
            % (here comparison with "machine zero" is done instead of
```

```
            % real zero)
    p1 = sum(M_full < 1e-10);

            % Make the same for tuning set
    M_full = (y_tune').*((w')*x_tune+b);
    p2 =  sum(M_full < 1e-10);

end
```

test_AS.m

```
function [z, b, p1, p2] = test_AS(mu)
    %% Preparations
        % Load the data
    [train,tune,test,dataDim] = getFederalistData;

        % Parse the data
    y = [train(:,1); tune(:,1)];
    y(y==2)=-1;
    x = [train(:,2:end); tune(:,2:end)]';

    %% Test
        % Prepare M and H matrices
    M = x(:,y==-1); % M is the set of objects of 1 class (Madison)
    H = x(:,y==1);  % H is the set of objects of 2 class (Hamilton)

        % Test by run_AS
    tic;
    [z,b,w,p1,p2] = run_AS(M,H,mu);
    time = toc;

        % Estimate train/tune errors
    error_train = 100*(p1 / 86);
    error_tune = 100*(p2 / 20);

        % Estimate w2
    w2 = w'*w;

        % Classify disputed papers
    c = (((w'*test'+b)>=0)*2-1)';

    %% Output testing results
    fprintf(1,'Train error: %2.2f %%\n',error_train);
    fprintf(1,'Tune error: %2.2f %%\n',error_tune);

    fprintf(1,'z= %f, b= %2.2f, ||w||2= %f, p1= %d, p2= %d\n',z,b,w2,p1,p2);

    fprintf(1,'Elapsed time: %f s\n',time);

    fprintf(1,'Disputed papers authorship prediction:\n');
    fprintf(1,'%d\n',c);
end
```

test_NOVEL.m

```
function [z, b, p1, p2] = test_NOVEL(mu)
    %% Preparations
        % Load the data
    [train,tune,test,dataDim] = getFederalistData;

        % Parse the data
    y = [train(:,1); tune(:,1)];
    y(y==2)=-1;
    x = [train(:,2:end); tune(:,2:end)]';

    %% Test
        % Prepare M and H matrices
    M = x(:,y==-1); % M is the set of objects of 1 class (Madison)
    H = x(:,y==1);  % H is the set of objects of 2 class (Hamilton)

        % test by run_NOVEL
    tic;
    [z,b,w,p1,p2] = run_NOVEL(M,H,mu);
    time = toc;

        % Estimate train/tune errors
    error_train = 100*(p1 / 86);
    error_tune = 100*(p2 / 20);

        % Estimate w2
    w2 = w'*w;

        % Classify disputed papers
    c = (((w'*test'+b)>=0)*2-1)';

    %% Output testing results
    fprintf(1,'Train error: %2.2f %%\n',error_train);
    fprintf(1,'Tune error: %2.2f %%\n',error_tune);

    fprintf(1,'z= %f, b= %2.2f, ||w||2= %f, p1= %d, p2= %d\n',z,b,w2,p1,p2);

    fprintf(1,'Elapsed time: %f s\n',time);

    fprintf(1,'Disputed papers authorship prediction:\n');
    fprintf(1,'%d\n',c);
end
```

test_quadprog.m

```
function [z, b, p1, p2] = test_quadprog(mu)
    %% Preparations
        % Load the data
    [train,tune,test,dataDim] = getFederalistData;

        % Parse the data
```

```
    y = [train(:,1); tune(:,1)];
    y(y==2)=-1;
    x = [train(:,2:end); tune(:,2:end)]';

    %% Solve it by quadprog
            % Prepare M and H matrices
    M = x(:,y==-1); % M is the set of objects of 1 class (Madison)
    H = x(:,y==1);  % H is the set of objects of 2 class (Hamilton)

    tic;
            % Parse the data (simulate same behavior as for AS and NOVEL
            % user defined functions)
                % training set
    x = [M(:,11:end) H(:,11:end)];
    y = [-ones(size(M(:,11:end),2),1); ones(size(H(:,11:end),2),1)];
            % tuning set (for tuning set - 10 points from M and 10 from H)
    x_tune = [M(:,1:10) H(:,1:10)];
    y_tune = [-ones(size(M(:,1:10),2),1); ones(size(H(:,1:10),2),1)];

            % Estimate C
    C = 1 / (mu * size(y,1));

        % Parse 'SVM' data into QP problem
    Q = x'*x;
    Y = diag(y);
    H = Y*Q*Y + 1e-6*eye(length(y));
    f = -ones(size(y));
    a = y';
    K = 0;
    Kl = zeros(size(y));
    Ku = C * ones(size(y));

        % Solve by quadprog
    options = optimset('Display','off');
    lamda = quadprog(H,f,[],[],a,K,Kl,Ku,[],options);

    w = x * (lamda.*y);

    e = 1e-6;
    ind = find(lamda > e & lamda < C-e);
    b = mean(y(ind) - x(:,ind)'*w);

            % To estimate objective function, at first
            % estimate indents M
    M_full = (y').*((w')*x+b);
            % Then errors
    E = max([(1 - M_full); zeros(size(y'))]);
            % Finally, z - objective function
    z = (1/length(y))*sum(E) + (mu/2)*(w'*w);
            % misclassified points - points with negative indent
            % (here comparison with "machine zero" is done instead of
            % real zero)
    p1 = sum(M_full < 1e-10);

            % Make the same for tuning set
```

```
    M_full = (y_tune').*((w')*x_tune+b);
    p2 =  sum(M_full < 1e-10);

    time = toc;

    error_train = 100*(1-sum(y==(((w'*x+b)>=0)*2-1)')/size(y,1));
    error_tune = 100*(1-sum(y_tune==(((w'*x_tune+b)>=0)*2-
1)')/size(y_tune,1));

            % Estimate w2
    w2 = w'*w;

            % Classify disputed papers
    c = (((w'*test'+b)>=0)*2-1)';
    %% Consume results


    fprintf(1,'Train error: %2.2f %%\n',error_train);
    fprintf(1,'Tune error: %2.2f %%\n',error_tune);

    fprintf(1,'z= %f, b= %2.2f, ||w||2= %f, p1= %d, p2= %d\n',z,b,w2,p1,p2);

    fprintf(1,'Elapsed time: %f s\n',time);

    fprintf(1,'Disputed papers authorship prediction:\n');
    fprintf(1,'%d\n',c);
end
```

Appendix C

Q3.m

```
close all;
clear all;
clc;

%% Preparations
    % Load the data
[train,tune,test,dataDim] = getFederalistData;

    % Parse the data
y = [train(:,1); tune(:,1)];
y(y==2)=-1;
x = [train(:,2:end); tune(:,2:end)]';

%% Test
    % Prepare M and H matrices
M = x(:,y==-1); % M is the set of objects of 1 class (Madison)
H = x(:,y==1);  % H is the set of objects of 2 class (Hamilton)

    % Specify mu value
mu = 0.1;
    % Define the array to store the tuning misclassification
    % (there are 20 points in tuning set, default value for p2 would 20)
p2_n = 20*ones(size(M,1),size(M,1));
z_n = 10000*ones(size(M,1),size(M,1));
for i=1:size(M,1)
    for j=(i+1):size(M,1)
            % Test by run_AS
        [z_n(i,j),b,w,p1,p2_n(i,j)] = run_NOVEL(M([i j],:),H([i j],:),mu);
    end
end


%% Output testing results
    % Estimate
j_min = ceil(find(p2_n==min(min(p2_n)),1)/70);
i_min = find(p2_n==min(min(p2_n)),1)-70*(j_min-1);

    % Draw tuning misclassifications as image
figure;
imagesc(p2_n);
hold on;
rectangle('Position',[(j_min-3) (i_min-3) 5 5], 'Curvature',[1
1],'EdgeColor','r','LineWidth',2);
xlabel('i');
ylabel('j');
title('Missclassified tuning points');

    % Output optimal features numbers and corresponding tuning
    % misclassifications
fprintf(1,'i= %d, j= %d, p2= %d\n',i_min, j_min, p2_n(i_min,j_min));
```

Q4.m

```
close all;
clear all;
clc;

%% Preparations
    % Load the data
[train,tune,test,dataDim] = getFederalistData;

    % Load the FederallistData again to get
    % the "wordlist"
load federalData;

    % Parse the data
y = [train(:,1); tune(:,1)];
y(y==2)=-1;
x = [train(:,2:end); tune(:,2:end)]';

    %
%% Test
    % Prepare M and H matrices
M = x(:,y==-1); % M is the set of objects of 1st class (Madison)
H = x(:,y==1);  % H is the set of objects of 2nd class (Hamilton)

    % Specify mu value
mu = 0.1;
    % Specify i and j - features (two features from all 70 features set)
i = 1;
j = 60;
    % Obtain optimal w,b SVM parameters
[z,b,w,p1,p2] = run_NOVEL(M([i j],:),H([i j],:),mu);



%% Plot resulting 2D "SVM mapping"

    % At first estimate new x,y limits
    % (to make figure a bit clearer)
x_min = min([M(i,:) H(i,:) test(:,i)']);
x_max = max([M(i,:) H(i,:) test(:,i)']);
x_width = x_max - x_min;
new_x_min = x_min - x_width*0.2;
new_x_max = x_max + x_width*0.2;

y_min = min([M(j,:) H(j,:) test(:,j)']);
y_max = max([M(j,:) H(j,:) test(:,j)']);
y_width = y_max - y_min;
new_y_min = y_min - y_width*0.2;
new_y_max = y_max + y_width*0.2;

figure;
hold on;
    % Plot all objects
    % ('o' - Hamilton, '+' - Medison and '*' - disputed)
```

```
scatter(M(i,:),M(j,:),'+');
scatter(H(i,:),H(j,:),'o');
scatter(test(:,i),test(:,j),'*');
    % Plot '2D-hyperplane - line'
line([new_x_min new_x_max],[-(b+new_x_min*(w(1)/w(2))) -
(b+new_x_max*(w(1)/w(2)))],'Color','r','LineWidth',2);
    % Set new limits for x,y
xlim([new_x_min new_x_max]);
ylim([new_y_min new_y_max]);
    % Set appropriate labels and title
xlabel(wordlist(i));
ylabel(wordlist(j));
title('SVM, 2D mapping');
    % Add legend
legend({'Madison','Hamilton','disputed','SVM hyperplane'});
```

Sources

[1] http://my.fit.edu/seces/slides/40.pdf

[2] https://people.csail.mit.edu/dsontag/courses/ml16/slides/lecture5.pdf

[3] https://www.cs.huji.ac.il/~shais/papers/ShalevSiSrCo10.pdf

[4] https://books.google.com.ua/books?id=3cBcAQAAQBAJ&dq=support+vector+machine+by+quadprog+matlab&hl=ru

[5] Glenn Fung / "The Disputed Federalist Papers: SVM Feature Selection via Concave Minimization"/ Journal of the ACM, Vol. V, No. N, Month 20YY