Εργασία 2 στη Μηχανική Μάθηση (1.5 μονάδες)

ΕΚΦΩΝΗΣΗ

Στην εργασία αυτή καλείστε να χρησιμοποιήσετε το σύστημα Weka για να μελετήσετε την καταλληλότητα διαφόρων μεθόδων μηχανικής μάθησης ως προς την εφαρμοσιμότητα και την καταλληλότητά τους για ένα συγκεκριμένο πρόβλημα ταξινόμησης. Το πρόβλημα αυτό είναι η ταξινόμηση μανιταριών σε δύο κατηγορίες, "βρώσιμα" ή "δηλητηριώδη", μέσω ενός συνόλου δεδομένων που θα βρείτε στο αρχείο mushroom.arff μέσα στο συμπιεσμένο αρχείο uci-20070111.tar.gz.

Αφού εγκαταστήσετε το Weka, εκτελείτε την GUI εκδοχή του, επιλέγετε "Explorer", "Open file", φορτώνετε το αρχείο δεδομένων (.arff) και πηγαίνετε στο tab "Classify". Με το "Choose" επιλέγετε τη μέθοδο που θέλετε να εφαρμόσετε. Οι μέθοδοι που πρέπει να διερευνήσετε είναι:
Δέντρα απόφασης (weka -> classifiers -> trees -> J48)
Πολυεπίπεδα νευρωνικά δίκτυα με σιγμοειδείς συναρτήσεις ενεργοποίησης (weka -> classifiers -> functions -> MultilayerPerceptron)
Μάθηση με στιγμιότυπα (weka -> classifiers -> lazy -> IBk)
Αφελής ταξινομητής Bayes (weka -> classifiers -> bayes -> NaiveBayes)
Μία μέθοδο της επιλογής σας από τις διαθέσιμες κάτω από το weka -> classifiers -> meta.

Μπορείτε σε πρώτη φάση, για να εξοικειωθείτε με το σύστημα, να χρησιμοποιήσετε ένα μικρότερο αρχείο δεδομένων, αντί του mushroom.arff, μέσα από το uci-20070111.tar.gz, π.χ. κάποιο από τα iris.arff ή vehicle.arff.

Για να αξιολογήσετε τις μεθόδους που εφαρμόσατε επάνω στα δεδομένα του mushroom.arff, να χρησιμοποιήσετε τη μέθοδο της διασταυρωμένης επικύρωσης 10 τμημάτων (10-fold cross validation). Για κάθε μέθοδο, θα πρέπει να πειραματιστείτε με τις τιμές των παραμέτρων της, αν υπάρχουν, ώστε να επιτύχετε το καλύτερο δυνατό ποσοστό σωστά ταξινομημένων στιγμιοτύπων.

Ως επιπλέον ερώτημα στην εργασία αυτή, με bonus προσαύξηση 5% στη συνεισφορά του βαθμού της στον τελικό βαθμό, επιλέξτε κάποιο από τα αρχεία δεδομένων που βρίσκονται μέσα στο συμπιεσμένο αρχείο 19MclassTextWc.zip, τα οποία αφορούν προβλήματα κατηγοριοποίησης κειμένων, και εφαρμόστε σε αυτό κάποια μέθοδο μηχανικής μάθησης από αυτές που διαθέτει το Weka, ώστε το ποσοστό των σωστά ταξινομημένων στιγμιοτύπων με τη μέθοδο της διασταυρωμένης επικύρωσης 10 τμημάτων να υπερβαίνει το 90%.

Το παραδοτέο για την εργασία είναι μία αναλυτική αναφορά που θα περιγράφει τη δουλειά σας και τα αποτελέσματά της. Στην αναφορά σας θα πρέπει να περιλαμβάνεται και μία λεπτομερής περιγραφή της 5ης μεθόδου που επιλέξατε εσείς να μελετήσετε. Προθεσμία παράδοσης είναι η ημέρα της γραπτής εξέτασης του μαθήματος κατά τη χειμερινή εξεταστική περίοδο.

ΤΕΧΝΙΚΑ ΣΤΟΙΧΕΙΑ

Όλα τα πειράματα έγιναν μέσω εφαρμογής που δημιούργησα σε java. Μέσω αυτής χρησιμοποίησα τη βιβλιοθήκη του weka ώπου δύναται να χρησιμοποιηθούν όλοι οι ταξινομητές που διαθέτει το weka. Οι λόγοι ήταν και οτι ήθελα να γράψω κώδικα που μου φαινόταν πιο ενδιαφέρον το οποίο είχε και πειραματισμό όμως και λόγω του ότι ήταν πιο εύκολο να τρέξεις πολλά πειράματα με διαφορετικές παραμέτρους σε σχέση με το *experimenter* mode του weka. Επίσης μέσω του κώδικά μου ήταν εφικτό να απομονώσω σε συνοπτική μορφή την πληροφορία που μου χρειαζόταν και να την ταξινομήσω ώστε να συγκρίνω τα αποτελέσματα. Ήταν κάτι επίσης που μου ήταν οικείο καθώς το είχα ξανακάνει για την πτυχιακή μου.

Τα πειράματα έγιναν σε μηχάνημα Ubuntu (22.04.3 LTS) x86_64 GNU/Linux
με memory 64GiB και cpu Intel(R) Core(TM) i7-10850H CPU @ 2.70GHz

Github: https://github.com/VicangelNik/machine_learning_ex2
IDE: IntelliJ
Language: java17

Επίσης χρειάστηκε να αυξήσω το heap size στα 40 GB ώστε να υπάρχει περισσότερη μνήμη για την παραλληλία στα πειράματα των ibk και SGD κυρίως όμως για τον multilayer perceptron.

Για τα formats του κώδικα, των errors και των weka configuration χρησιμοποίησα την προέκταση του google doc με όνομα code blocks . Χρησιμοποιήθηκε η γλώσσα fortran με θέματα τα atelier-cave-dark, paraiso-light, atelier-cave-light για κώδικα, weka configurations και σφάλματα και αντίστοιχα.

ΠΕΙΡΑΜΑΤΑ - ΑΠΟΤΕΛΕΣΜΑΤΑ

Όλα τα αποτελέσματα των αλγορίθμων που χρησιμοποιήθηκαν παραθέτονται σε αρχεία με τα αντίστοιχα ονόματα. Ο Naive Bayes είχε ουσιαστικά ένα πείραμα. Όμως οι υπόλοιποι αλγόριθμοι μηχανικής μάθησης είχαν πάρα πολλά πειράματα για αυτό και μορφοποίησα το αποτέλεσμα που καταγράφεται με την πιο σημαντική πληροφορία ώστε τα αρχεία αποτελεσμάτων να είναι λιτά και περιεκτικά. Επίσης ταξινόμησα τα αποτελέσματα βάσει των χαρακτηριστικών τους προτού γραφούν σε αρχείο για καλύτερη σύγκριση. Μια μικρή υποσημείωση ως προς τους χρόνους δημιουργίας των μοντέλων, οι χρόνοι που καταγράφονται στα αρχεία ενδεχομένως να μην είναι αντιπροσωπευτικοί αν έτρεχε κάθε πείραμα μεμονωμένα καθώς παρατήρησα ότι σε αρκετές περιπτώσεις λόγω της παραλληλίας του τρεξίματος των πειραμάτων υπήρχε διαφορά στους αναγραφόμενους χρόνους από ότι όταν έτρεχαν μεμονωμένα είτε και μέσω του weka. Χρησιμοποίησα παραλληλία στη java ώστε συνολικά να τρέξουν περισσότερα πειράματα στον ίδιο χρόνο.

NAIVE BAYES

Ο Naive bayes αλγοριθμος δεν έχει παραμέτρους που θα μπορούσαν να χρησιμοποιηθούν. Οι 2 παράμετροι που έχει δεν έχουν κάποια επίδραση σε “ονομαστικά” (nominal) δεδομένα όπως το mushroom.arff. Το μοντέλο καθώς και ο χρόνος εκπαίδευσης χρειάζονται ελάχιστο χρόνο να περαττωθούν. Ο αλγόριθμος επίσης δίνει ποσοστό σωστά ταξινομημένων στιγμιοτύπων πάνω από 95%.

```
Correctly Classified Instances        7785               95.8272 %
Incorrectly Classified Instances       339                4.1728 %
Kappa statistic                          0.9162
Mean absolute error                      0.0419
Root mean squared error                  0.1757
Relative absolute error                  8.397  %
Root relative squared error             35.1617 %
Total Number of Instances             8124


=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0.992    0.078    0.932      0.992    0.961      0.918    0.998     0.998     e
               0.922    0.008    0.991      0.922    0.955      0.918    0.998     0.998     p
Weighted Avg.    0.958    0.044    0.960      0.958    0.958      0.918    0.998     0.998
```

```
=== Confusion Matrix ===

   a    b   <-- classified as
4176   32 |    a = e
 307 3609 |    b = p
```

C4.5 (J48)

Ουσιαστικά ο J48 είναι αλγόριθμος της μηχανικής μάθησης δέντρων απόφασης και ανάλογα με μία παράμετρο του (reducedErrorPruning, -R) λειτουργεί είτε ως C4.5 ή ως Reduced Error Pruning. Γενικά, το να δημιουργηθεί το μοντέλο και ο χρόνος εκπαίδευσης που χρειάζονται οι 2 αυτοί αλγόριθμοι του J48 είναι μικρός, συγκρίσιμος του naive bayes. Αυτό που παρατηρώ, από τα αποτελέσματα είναι ότι ο C4.5 με τις κατάλληλες παραμέτρους πετυχαίνει σε περίπου τον ίδιο χρόνο (λίγα δέκατα του δευτερολέπτου παραπάνω) με τον Reduced Error Pruning ποσοστό σωστά ταξινομημένων στιγμιοτύπων πάνω από 99.9 % με λιγότερα φύλλα και συνολικά μικρότερο μέγεθος του δέντρου. Οπότε αν μας ενδιέφερε να πετυχαίνουμε, με ποσοστό σωστών ταξινομημένων στιγμιοτύπων περίπου 99.9 %, μικρή χωρητικότητα (μνήμη) που καταλαμβάνει το μοντέλο θα επέλεγα τον C4.5 ενώ αν μας ενδιέφερε ο χρόνος θα επέλεγα Reduced Error Pruning . Όμως, Ο C.45 πετυχαίνει πολλές περισσότερες φορές ποσοστό απόλυτα σωστών ταξινομημένων στιγμιοτύπων (100%) σε σχέση με τον αλγόριθμο reduced error pruning και με μικρότερο μέγεθος του μοντέλου (δέντρου). **Έτσι συνολικά θα επέλεγα τον C.45 ως τον αλγόριθμο που τα πηγαίνει καλύτερα στο συγκεκριμένο dataset (mushroom.arff)**. Παρακάτω παρατίθενται ένα πείραμα με το ίδιο (ανώτερο) ποσοστό σωστά ταξινομημένων στιγμιοτύπων για κάθε αλγόριθμο του J48.

C4.5

Στον C.45 χρησιμοποιήθηκαν οι τιμές του παράγοντα αυτοπεποίθησης εύρους από 0.05 - 0.5 με βήμα 0.05. Επίσης χρησιμοποίηθηκαν διάφορες παράμετροι εναλλάξ και για όλους τους συνδυασμούς η MDL διόρθωση και pruning (κλάδεμα των μερών του δέντρου που δεν βοηθούν στο βέλτιστο αποτέλεσμα). Συνολικά μέσω του προγράμματός μου έτρεξαν 3199 πειράματα. Ο C.45 πετυχαίνει πολλές περισσότερες φορές ποσοστό απόλυτα σωστών ταξινομημένων στιγμιοτύπων (100%) σε σχέση με τον αλγόριθμο reduced error pruning.

Το παρακάτω αποτέλεσμα προκύπτει από διάφορους συνδυασμούς παραμέτρων όπως:

```
Options: -O -B -J -A -M 4 -C 0.05
```

```
Number of Leaves  :     9

Size of the tree :      17

Correctly Classified Instances        8124               100      %
Incorrectly Classified Instances         0                 0      %
Kappa statistic                          1
Mean absolute error                      0
Root mean squared error                  0
Relative absolute error                  0      %
Root relative squared error              0      %
Total Number of Instances             8124

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC
ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000
1.000     1.000     e
                 1.000    0.000    1.000      1.000    1.000      1.000
1.000     1.000     p
Weighted Avg.    1.000    0.000    1.000      1.000    1.000      1.000
1.000     1.000

=== Confusion Matrix ===

    a    b   <-- classified as
 4208    0 |    a = e
    0 3916 |    b = p
```

Reduced Error Pruning

Στον reduced error pruning αλγόριθμο χρησιμοποιήθηκαν οι παράμετροι όπως ο ελάχιστος αριθμός στιγμιότυπων σε ένα φύλλο 1 έως 5 καθώς και ο αριθμός των δεδομένων που θα χρησιμοποιηθούν για pruning από 2 έως 6. Επίσης χρησιμοποίηθηκαν διάφορες παράμετροι εναλλάξ και για όλους τους συνδυασμούς όπως MDL και Laplace διορθώσεις. Συνολικά μέσω του προγράμματός μου έτρεξαν 3994 πειράματα.

Το παρακάτω αποτέλεσμα προκύπτει από διάφορους συνδυασμούς παραμέτρων όπως:

```
Options: -R -O -B -J -A -S -N 3 -M 4 -Q 1
```

```
Number of Leaves  :     25

Size of the tree :      30


Correctly Classified Instances        8124               100      %
Incorrectly Classified Instances         0                 0      %
Kappa statistic                          1
Mean absolute error                      0
Root mean squared error                  0
Relative absolute error                  0      %
Root relative squared error              0      %
Total Number of Instances             8124

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC
ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000
1.000     1.000     e
                 1.000    0.000    1.000      1.000    1.000      1.000
1.000     1.000     p
Weighted Avg.    1.000    0.000    1.000      1.000    1.000      1.000
1.000     1.000

=== Confusion Matrix ===

    a    b   <-- classified as
 4208    0 |    a = e
    0 3916 |    b = p
```

K-nearest neighbours (IBK)

Γενικά, ο χρόνος που απαιτούσε η δημιουργία του μοντέλου και ο χρόνος εκπαίδευσης στον αλγόριθμο του πλησιέστερου γείτονα ήταν σχετικά μικρός που όμως άλλαζε εμφανώς λόγω των παραμέτρων του. Σίγουρα συνολικά πιο αργός από J48 και Naive Bayes. **Κυρίως αυτό που άλλαζε ήταν ο χρόνος που απαιτούνταν για να ολοκληρωθεί η διασταυρωμένη επικύρωση 10 τμημάτων σε σχέση με τους j48 και naive bayes.** Στο πρόγραμμα πειραμάτων που δημιούργησα έτρεξαν σε παραλληλία 1,488 πειράματα. Στα πειράματα δοκίμασα κυρίως την αναζήτηση brute force με διάφορους συνδυασμούς μετρικών απόστασης(Euclidean, Manhattan, Chebyshev, Minkowski). Επίσης, χρησιμοποιήθηκαν από τον 1 έως και τους 30 πλησιέστερους γείτονες για την ταξινόμηση καθώς και κάποια βάρη στους γείτονες αυτούς. Μέσω του weka application δοκίμασα πολύ συνοπτικά και τους υπόλοιπους αλγορίθμους αναζήτησης (KDTree, CoverTree, BallTree). Οι μέθοδοι αυτοί αναζήτησης δεν μπορούσαν να διαχειριστούν τις απούσες τιμές στη mushroom βάση. Οπότε το weka παρέχει και έναν άλλο αλγόριθμο αναζήτησης ονόματι “FilteredNeighbourSearch”. Σε αυτόν μπορείς να θέσεις εκτός της μεθοδου αναζήτησης (KDTree, CoverTree, BallTree, BruteForce) και μεταξύ άλλων και φίλτρα στα δεδομένα. Έτσι το weka παρέχει φίλτρο “replaceMissingValues” και έτσι δοκίμασα όλους τους αλγορίθμους αναζήτησης. Παρακάτω παρατίθεται κάποια αποτελέσματα αντιπροσωπευτικά της απόδοσης του αλγορίθμου.

```
weka.classifiers.lazy.IBk -K 1 -W 0 -X -A
"weka.core.neighboursearch.LinearNNSearch -A
\"weka.core.ManhattanDistance -R first-last\""
```

```
Correctly Classified Instances        8124               100      %
Incorrectly Classified Instances         0                 0      %
Kappa statistic                          1
Mean absolute error                      0
Root mean squared error                  0
Relative absolute error                  0.0029 %
Root relative squared error              0.003  %
Total Number of Instances             8124

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC
ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000
1.000     1.000     e
                 1.000    0.000    1.000      1.000    1.000      1.000
1.000     1.000     p
Weighted Avg.    1.000    0.000    1.000      1.000    1.000      1.000
1.000     1.000

=== Confusion Matrix ===
```

```
    a    b   <-- classified as
 4208    0 |    a = e
    0 3916 |    b = p
```

```
 weka.classifiers.lazy.IBk -K 1 -W 0 -X -A
"weka.core.neighboursearch.FilteredNeighbourSearch -F
\"weka.filters.unsupervised.attribute.ReplaceMissingValues \" -S
\"weka.core.neighboursearch.BallTree -A \\\"weka.core.EuclideanDistance
-R first-last\\\" -C
\\\"weka.core.neighboursearch.balltrees.TopDownConstructor -S
weka.core.neighboursearch.balltrees.PointsClosestToFurthestChildren -N
40\\\"\""
```

```
Correctly Classified Instances        8124               100      %
Incorrectly Classified Instances         0                 0      %
Kappa statistic                          1
Mean absolute error                      0
Root mean squared error                  0
Relative absolute error                  0.0029 %
Root relative squared error              0.003  %
Total Number of Instances             8124

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC
ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000
1.000     1.000     e
                 1.000    0.000    1.000      1.000    1.000      1.000
1.000     1.000     p
Weighted Avg.    1.000    0.000    1.000      1.000    1.000      1.000
1.000     1.000

=== Confusion Matrix ===

    a    b   <-- classified as
 4208    0 |    a = e
    0 3916 |    b = p
```

Για κάποιο λόγο που δεν βρήκα και είχε να κάνει με τον κώδικα της weka βιβλιοθήκης που χρησιμοποίησα, το πρόγραμμα παρουσίαζε σφάλμα με τις παρακάτω παραμέτρους, σε ένα μόνο πείραμα:

```
weka.classifiers.lazy.IBk -K 30 -W 0 -F -A
"weka.core.neighboursearch.LinearNNSearch -A
\"weka.core.ManhattanDistance -R first-last\" -P"
```

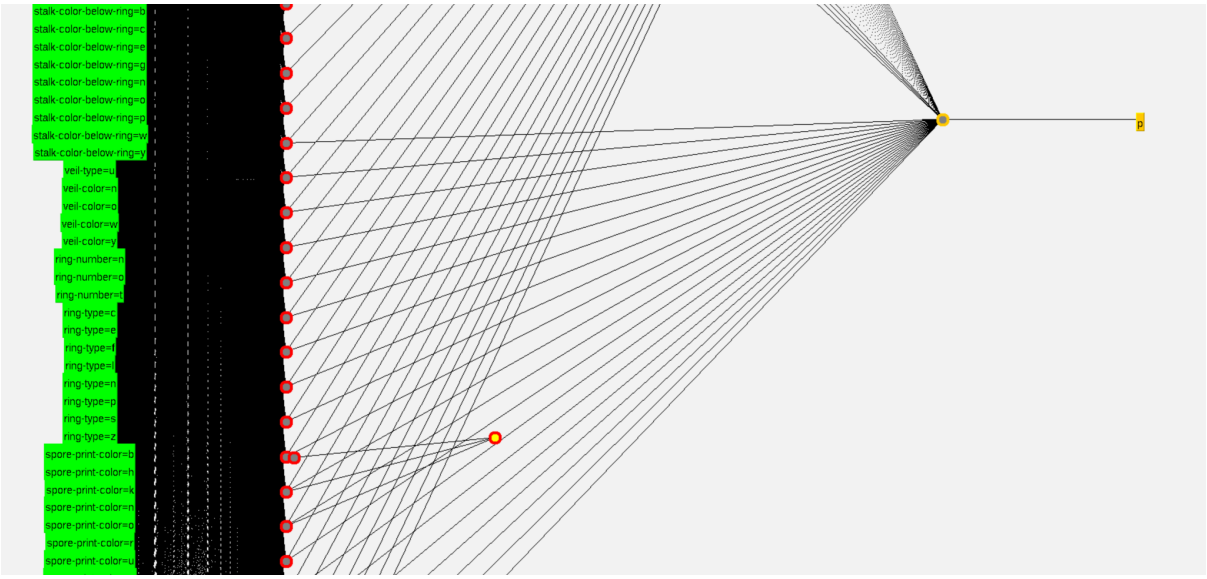
```
java.lang.RuntimeException: java.lang.ArrayIndexOutOfBoundsException:
Index 506 out of bounds for length 501
```

Αλλά στο weka έτρεξε σωστά με αποτέλεσμα:

```
Correctly Classified Instances        8067               99.9876 %
Incorrectly Classified Instances         0                0      %
Kappa statistic                          1
Mean absolute error                      0.0005
Root mean squared error                  0.0082
Relative absolute error                  0.1098 %
Root relative squared error              1.6489 %
UnClassified Instances                   1                0.0124 %
Total Number of Instances             8068

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC
ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000
1.000     1.000     e
                 1.000    0.000    1.000      1.000    1.000      1.000
1.000     1.000     p
Weighted Avg.    1.000    0.000    1.000      1.000    1.000      1.000
1.000     1.000

=== Confusion Matrix ===

    a    b   <-- classified as
 4193    0 |    a = e
    0 3874 |    b = p
```

Multilayer Perceptron

Ο πολυεπίπεδος νευρώνας (Multilayer perceptron) σε όλα τα πειράματα που έτρεξα είχε απόδοση στο 100%. Δηλαδή είχε ποσοστό απόλυτα σωστών ταξινομημένων στιγμιοτύπων (100%). Όμως ήταν απελπιστικά αργός ειδικά όταν ο αριθμός των εποχών αυξανόταν. Έτσι αναγκάστηκα να κόψω τα πειράματα και ενώ ήταν να τρέξω 1,215 το σταμάτησα στα 319. Τα πειράματα που δοκίμασα από το πρόγραμμα περιελάμβαναν αριθμό εποχών (300, 500, 700) καθώς επίσης σαν ρυθμούς εκμάθησης και momentum (0.1, 0.2, 0.3). Επίσης δοκίμασα αν επηρεάζουν τη απόφαση και άλλες παράμετροι όπως το ανώτατο κατώφλι επιτρεπτού σφάλματος στην επικύρωση και το ποσοστό των δεδομένων που θα χρησιμοποιηθούν προς επικύρωση χωρίς κάποια διαφορά. Δοκίμασα επίσης με διάφορες παραμέτρους και από το εργαλείο του weka. Αυτό που μου άρεσε στο weka είναι το γραφικό περιβάλλον στο multilayer perceptron. Σου δίνει τη δυνατότητα να αλλάξεις τα learning rate και momentum rates ανά εποχή ενώ επίσης μπορείς να φτιάξεις τη δική σου αρχιτεκτονική ως προς τα επίπεδα και τις συνδέσεις, να βάλεις επιπρόσθετους κόμβους και να τους συνδέσεις ή να τους αφαιρέσεις. Όπως στην παρακάτω εικόνα.



Δοκίμασα επίσης με κάποια πιο ακραία σενάρια όπως με 0 ενδιάμεσα επίπεδα (hidden layers). Μετά από διάφορους συνδυασμούς βρήκα ότι για τις προκαθορισμένες παραμέτρους και για μόλις 10 εποχές εκπαίδευσης, δηλαδή της εφαρμογής του backpropagation στο νευρωνικό δίκτυο, με 0 ενδιάμεσα επίπεδα το ποσοστό σωστών ταξινομημένων στιγμιοτύπων ήταν (100%). Αν άλλαζες το learning rate αυτό άλλαζε δραστικά και το αποτέλεσμα. **Το γεγονός ότι με 0 ενδιάμεσα επίπεδα το ποσοστό σωστών ταξινομημένων στιγμιοτύπων είναι υψηλό που φτάνει και το 100% μας δείχνει ότι τα δεδομένα του dataset είναι γραμμικώς διαχωρίσιμα**. Επίσης, όταν ο αριθμός των ενδιάμεσων επιπέδων είναι μικρός οι υπολογιστικοί πόροι σε μνήμη που χρειάζονται είναι λίγοι και όταν ο αριθμός των εποχών είναι μικρός βοηθάει και στη μνήμη που χρειάζεται αλλά και στο χρόνο δημιουργίας και εκπαίδευσης του μοντέλου. Ο multilayer perceptron σίγουρα είναι πολύ αργός όμως με 0 ενδιάμεσα επίπεδα χρονικά ανεκτός ίσως συγκρίσιμος σε κάποιες περιπτώσεις με τον IBK.

```
weka.classifiers.functions.MultilayerPerceptron -L 0.4 -M 0.2 -N 9 -V 0
-S 0 -E 20 -H 0 -R
```

```
Time taken to build model: 0.28 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        8124               100      %
Incorrectly Classified Instances         0                 0      %
Kappa statistic                          1
Mean absolute error                      0.0035
Root mean squared error                  0.0098
Relative absolute error                  0.6967 %
Root relative squared error              1.9617 %
Total Number of Instances             8124

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC
ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000
1.000     1.000     e
                 1.000    0.000    1.000      1.000    1.000      1.000
1.000     1.000     p
Weighted Avg.    1.000    0.000    1.000      1.000    1.000      1.000
1.000     1.000

=== Confusion Matrix ===

    a    b   <-- classified as
 4208    0 |    a = e
    0 3916 |    b = p
```

Stochastic Gradient Descent (SGD)

Η μέθοδος Στοχαστικής Σταδιακής Καθόδου (Stochastic Gradient Descent) εξηγείται παρακάτω. Γενικά, η ταχύτητα δημιουργίας του μοντέλου και εκπαίδευσης του είναι συγκρίσιμη με τον ibk και λίγο καλύτερη αναλόγως των παραμέτρων του ibk. Για όλα τα πειράματα που έτρεξα, σύνολο 108 το ποσοστό σωστών ταξινομημένων στιγμιοτύπων είναι 100%. Ο αλγόριθμος έχει άριστη απόδοση και σε καλό χρόνο. Ως ρυθμούς εκμάθησης δοκίμασα τις τιμές (0.005, 0.01, 0.02). Ενώ το λ το εξέτασα για τις τιμές 1.0E-3, 1.0E-4, 1.0E-5.

Περιγραφή Μεθόδου

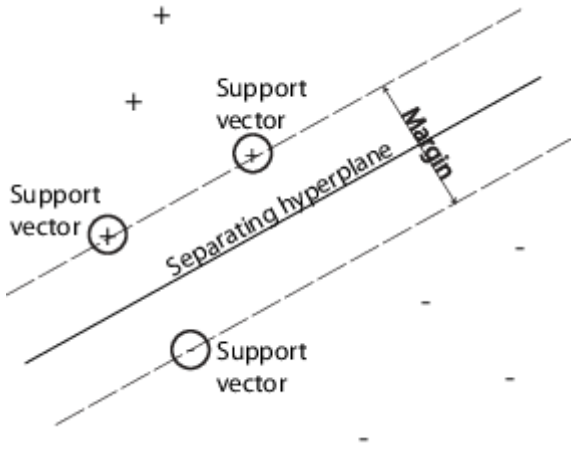
Ο gradient descent αλγόριθμος γενικά, είναι ένας αλγόριθμος βελτιστοποίησης που με επαναληπτική διαδικασία στοχεύει στην ελαχιστοποίηση της συνάρτησης κόστους. Δηλαδή της συνάρτησης αυτής που ποσοτικοποιεί τη διαφορά μεταξύ των τιμών που προβλέφθηκαν από το μοντέλο και των πραγματικών τιμών στόχων. Με άλλα λόγια, η συνάρτηση κόστους ποσοτικοποιεί το πόσο καλά αποδίδει ο αλγόριθμος, στην προκειμένη περίπτωση στην απόφαση της ταξινόμησης μεταξύ 2 ενδεχομένων, αν τα μανιτάρια είναι βρώσιμα ή δηλητηριώδη. Ο αλγόριθμος αυτός ουσιαστικά ελαχιστοποιώντας την συνάρτηση κόστους βρίσκει τις τιμές των παραμέτρων του μοντέλου μηχανικής μάθησης που ελαχιστοποιούν αυτή τη συνάρτηση κόστους. Ελαχιστοποιώντας τη συνάρτηση κόστους κάνουμε το μοντέλο μας πιο ακριβή στις προβλέψεις του ή και το κάνουμε να αποδίδει καλύτερα σε καινούργια δεδομένα.

Ο αλγόριθμος που χρησιμοποιεί το weka είναι ο mini-batch stochastic gradient descent παραλλαγή του gradient descent. Επίσης, για το πρόβλημα της ταξινόμησης ο αλγόριθμος αυτός του weka ως συνάρτηση κόστους χρησιμοποιεί τη Hinge Loss (SVM).

Ο gradient descent αλγόριθμος για να επικαιροποιήσει της παραμέτρους του μοντέλου υπολογίζει τον ρυθμό επικαιροποίησης (την τιμή η οποία θα αλλάξει τις παραμέτρους του μοντέλου) υπολογίζοντας τη συνάρτηση κόστους για όλα τα δεδομένα εκπαίδευσης. Αυτό τον κάνει χρονικά αργό και κοστοβόρο από άποψη πόρων. Παρουσιάζει μια σταθερή σύγκλιση και γενικά καταλήγει να βρίσκει το ολικό ελάχιστο καθώς επίσης είναι ντετερμινιστικός αφου όσες φορές και αν τον τρέξεις θα έχει ίδιο ρυθμό σύγκλισης.
Για να μετριαστεί το υπολογιστικό κόστος που χρειάζεται ο gradient descent εισήχθη ο stochastic gradient descent που αντί για όλο το dataset χρησιμοποιεί μια συνάρτηση κόστους από ένα τυχαία επιλεγμένο στιγμιότυπο για να υπολογίσει τον ρυθμό επικαιροποίησης. Όμως τον κάνει στοχαστικό, δηλαδή αν τρέξεις ξανά τον αλγόριθμο δεν θα έχει ποτέ ίδιο ρυθμό επικαιροποίησης και κάνει ασταθή τη σύγκλιση η οποία δεν καταλήγει απαραίτητα στο ολικό ελάχιστο.
Για να βρεθεί η χρυσή τομή, δημοφιλής αλγόριθμος είναι ο mini-batch stochastic gradient descent ο οποίος λαμβάνει υπόψη του όχι ένα στιγμιότυπο ούτε όλο το dataset παρά έναν αριθμό στιγμιοτύπων (batch) για τον υπολογισμό του ρυθμού επικαιροποίησης. Αυτό τον κάνει γενικά έναν αλγόριθμο που πετυχαίνει μια ισορροπία μεταξύ των υπέρ και κατά των gradient descent και stochastic.

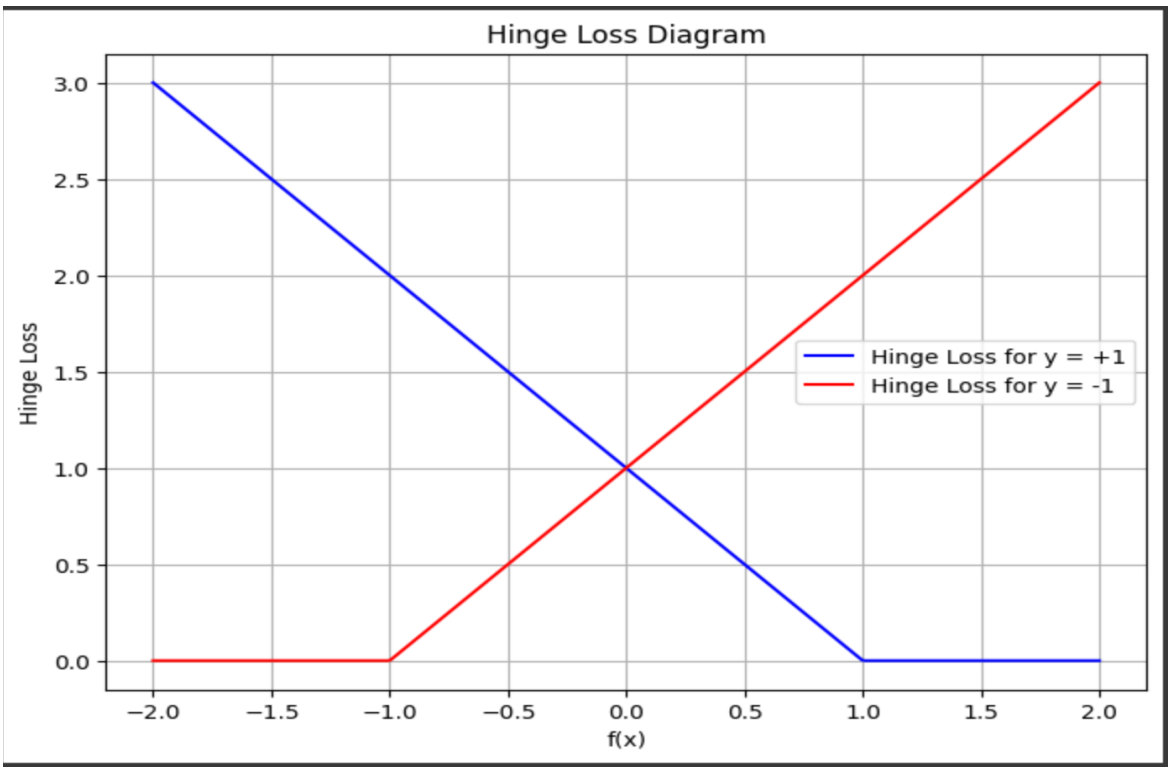
Στην πραγματικότητα λοιπόν ο αλγόριθμος ταξινόμησης που χρησιμοποιεί το weka είναι οι μηχανές διανυσμάτων στήριξης (SVM) ευέλικτης απόστασης (soft margin) με συνάρτηση κόστους την Hinge Loss όπου αυτή η συνάρτηση κόστους βελτιστοποιείται μέσω του stochastic gradient descent. Μάλιστα ο αλγόριθμος όπως αναφέρει και το javadoc του weka μετατρέπει τα δεδομένα από ονομαστικά σε δυαδικά και τα κανονικοποιεί.

Με απλά λόγια οι μηχανές διανυσμάτων στήριξης ταξινομούν τα άγνωστα σημεία σύμφωνα με την πλευρά του υπερ επίπεδου στην οποία βρίσκονται. Τα διανύσματα τα οποία ορίζουν το υπερεπίπεδο που χωρίζει τις δύο τάξεις ονομάζονται διανύσματα υποστήριξης (support vectors)



Για την ταξινόμηση αυτή και την εκπαίδευση του μοντέλου χρησιμοποιείται η συνάρτηση κόστους Hinge Loss. Η συνάρτηση αυτή
$L(y) = max(0, 1 - t \cdot y)$, t = πραγματική τιμή και y = πρόβλεψη (το αποτέλεσμα του ταξινομητή) τιμωρεί τα δεδομένα όταν L(y) > 1 το οποίο είναι σε αντιστοιχία με την ιδέα πίσω από τα SVM με soft margin. Δηλαδή αφήνεται ένα ευέλικτο περιθώριο στην απόφαση του ταξινομητή εφόσον η η ταξινόμηση τελικά κατατάσει σωστά τα δεδομένα στη θετική (1) ή αρνητική (-1) αλλά ενδιάμεσα των support vector και separating hyperplane.

Για παράδειγμα αν η πραγματική τιμή είναι 1 και η πρόβλεψη 0.5 το αποτέλεσμα της Hinge Loss θα είναι $L(y) = max(0, 1 - 1 * 0.5) = 0.5$.
Όμως αν η πραγματική τιμή είναι -1 και η πρόβλεψη 0.5 το αποτέλεσμα της Hinge Loss θα είναι $L(y) = max(0, 1 - (-1) * 0.5) = 1.5$.
Αντίστοιχα, αν η πραγματική τιμή είναι +1 και η πρόβλεψη 1.5 δηλαδή σωστή πρόβλεψη, το αποτέλεσμα της Hinge Loss θα είναι $L(y) = max(0, 1 - 1 * 1.5) = 0$. Άρα 0 κόστος.



Έτσι επιτυγχάνεται ο αντικειμενικός στόχος του SVM δηλαδή, η τοποθέτηση του υπερ επιπέδου στο χώρο έτσι ώστε να επιτευχθεί η μέγιστη διαχωρισιμότητα μεταξύ των 2 κλάσεων. Η ιδέα αυτή, είναι κάτι που αντιτίθεται σε άλλες μεθόδους όπως η γραμμική παλινδρόμηση που αντικειμενικός τους στόχος είναι η ελαχιστοποίηση της απόστασης μεταξύ των δεδομένων.

Η αντικειμενική συνάρτηση που περιγράφει τα SVM είναι η:

$J(w) = 1/2 * w^T * w + C * \Sigma_i max(0, 1\, y_i(w^T * x_i + b))$

Το πρώτο σκέλος πριν το + $1/2 * w^T * w$ λέγεται regularization term και μεγιστοποιεί την απόσταση μεταξύ των svms καθώς επίσης προωθεί την καλύτερη γενίκευση του μοντέλου. Το C είναι μια υπερ παράμετρος του μοντέλου που εξισορροπεί την εναλλαγή μεταξύ της μεγάλης απόστασης των svm και μικρού hinge loss.

Το τρίτο μέρος $\Sigma_i max(0, 1\, y_i(w^T * x_i + b))$ το hinge loss γραμμένο με τα βάρη w και κάποια προστιθέμενη προκατάληψη (bias) b. Πριν γράφτηκε πιο απλουστευμένα $L(y) = max(0, 1 - t \cdot y)$ για να αναλυθεί.

Σε αυτή τη συνάρτηση εφαρμόζεται από το weka ο mini-batch stochastic gradient descent που στόχο έχει να ελαχιστοποιήσει την αντικειμενική συνάρτηση

$min_{w,b}(1/2 * w^T * w + C * N\, max(0, 1\, y_i(w^T * x_i + b))$

Το Σ παραπάνω συμβολίζει οτι η συνάρτηση διατρέχει όλο το dataset όμως στον mini batch stochastic gradient descent αντικαθιστάται με το N που συμβολίζει τον αριθμό των δεδομένων υποσύνολο όλου του dataset (batch)

ώπου για τις εποχές από 1 - T εφαρμόζει επαναληπτικά

1) Παίρνει ένα τυχαίο υποσύνολο των δεδομένων από το dataset x_i, y_i
2) Υπολογίζει την παράγωγο για το υποσύνολο των δεδομένων ως προς w $J^T(w) = \nabla J^T(w^{t+1})$
3) Επικαιροποιεί το w στη συνάρτηση, το νέο w: $w^t = w^{t+1} - \gamma_t \nabla J^T(w^{t+1})$

Τελικώς επιστρέφεται το w που ελαχιστοποιεί τη συνάρτηση. Ο αλγόριθμος είναι εγγυημένο οτι συγκλίνει στο ολικό ελάχιστο αν το γ_t είναι αρκετά μικρό. Παράμετρος που εξυπηρετεί αυτό το σκοπό καθώς μόνος του ο stochastic gradient descent θα φτάσει σε ελάχιστο όχι απαραίτητα ολικό.

Από weka javadoc

```
Implements stochastic gradient descent for learning various linear
models (binary class SVM, binary class logistic regression, squared
loss, Huber loss and epsilon-insensitive loss linear regression).
Globally replaces all missing values and transforms nominal attributes
into binary ones. It also normalizes all attributes, so the coefficients
in the output are based on the normalized data.
For numeric class attributes, the squared, Huber or epsilon-insensitve
loss function must be used. Epsilon-insensitive and Huber loss may
require a much higher learning rate.
```

ΣΥΜΠΕΡΑΣΜΑΤΑ

Το dataset που αξιολογήθηκε είναι το mushroom.arff το οποίο περιέχει 8124 στιγμιότυπα μανιταριών με 22 χαρακτηριστικά και 1 ακόμα χαρακτηριστικό αν είναι βρώσιμα ή δηλητηριώδη που ως προς αυτό έγινε η ταξινόμηση. Οι αλγόριθμοι ταξινόμησης που αξιολογήθηκαν όπως ονομάζονται από το weka είναι οι naive bayes, j48, ibk, multilayer perceptron και sgd. Ο j48 και στα 2 modes του ειδικά όμως με τον προκαθορισμένο του j48 τον C.45 έδειξε να τα πηγαίνει καλύτερα χρονικά και να πετυχαίνει με πολλές διαφορετικές παραμέτρους απόλυτο ποσοστό σωστά ταξινομημένων στιγμιοτύπων. Γενικά είχε την καλύτερη απόδοση. Οι ibk και sgd με τον δεύτερο λίγο πιο αργό χρονικά είχαν επίσης εξαιρετική απόδοση για πολλές διαφορετικές παραμέτρους. Ο multilayer perceptron έδειξε να είναι εξαιρετικά αργός αλγόριθμος και να χρειάζεται και αρκετούς επεξεργαστικούς πόρους καθώς και μνήμης. Το γεγονός πάντως ότι μπορούσε και με κανένα ενδιάμεσο επίπεδο να πετύχει μέγιστη απόδοση δείχνει τη γραμμική σχέση των δεδομένων. Όταν χρειάζεται λίγα επίπεδα αλλά και μικρό αριθμό κύκλων εκπαίδευσης μέσω του αλγορίθμου της οπισθοδιάδοσης (εποχές) είναι σημαντικά πιο γρήγορος. Κάτι άλλο που είναι γνωστό και επιβεβαιώνει γιατί τα δέντρα απόφασης τα πηγαίνουν καλύτερα είναι το γεγονός ότι με μόλις 4 κανόνες που περιλαμβάνουν μόλις 6 χαρακτηριστικά μπορούν και πετυχαίνουν ποσοστό σωστής ταξινόμησης στο 100%. Τα δεντρα απόφασης έχουν αρκετές ομοιότητες και στην γενικότερη ιδέα τους είναι ίδιοι με τους rule-based αλγορίθμους. Δοκίμασα μερικούς rule-based αλγορίθμους όπως PART και JRip και είχαν εξαιρετική απόδοση και ταχύτητα. Αφού λοιπόν μπορεί το dataset να ταξινομηθεί με απόλυτη επιτυχία με 4 απλούς κανόνες μας δείχνει και τη γραμμικότητα της συσχέτισης μεταξύ των χαρακτηριστικών του κάτι που το είχαμε καταλάβει και μέσω της προηγούμενης διαπίστωσης με τον multilayer perceptron.

BONUS

Επέλεξα το αρχείο **wap.wc.arff** το οποίο περιέχει 1560 στιγμιότυπα με 8460 χαρακτηριστικά. Το dataset αυτό περιέχει πολλά χαρακτηριστικά (8460) που τους δίνει μόνο αριθμητικές τιμές και ο ταξινομητής καλείται να κατατάξει τα στιγμιότυπα αυτά σε 20 κλάσεις. Δηλαδή αν εντάσονται νοηματικά σε έαν από τα ακόλουθα:
Art,Business,Cable,Culture,Entertainment,Film,Health,Industry,Media,Multimedia,Music,Online,People,Politics,Review,Sports,Stage,Technology,Television,Variety

Το dataset αυτό υπάρχει στο openml - ένα χώρο που παρέχει δωρεάν διάφορα εργαλεία και πάρα πολλά dataset για machine learning χρήση.

Δοκίμασα διάφορους αλγόριθμους όπως δέντρα (Hoeffding Tree, Decision Stump) και rule-based αλγορίθμους όπως PART, Jrip, Naive Bayes Multinomial χωρίς καλή απόδοση. Η συσχέτιση μεταξύ των χαρακτηριστικών είναι περίπλοκη και μάλλον όχι γραμμική. Επίσης μιλάμε για 8460 χαρακτηριστικά-διαστάσεις που είναι σίγουρα αρκετά. Σίγουρα όμως λίγα αν είχαμε να κάνουμε με κάποιο, για παράδειγμα βιολογικό dataset.

Δοκίμασα όσους περισσότερους αλγόριθμους μπορούσα και με παραλλαγές.

Οι NaiveBayesMultinomial και NaiveBayesMultinomialUpdatable που επί της ουσίας είναι ίδιοι καταφέρνουν ποσοστό 81.0897%.

Ο John Platt's sequential minimal optimization algorithm (SMO) 82.1795 %
για παραμέτρους

```
weka.classifiers.functions.SMO -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W
1 -K "weka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C
250007" -calibrator "weka.classifiers.functions.GaussianProcesses -L 1.0
-N 0 -K \"weka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C
250007\" -S 1"
```

Ο Locally weighted learning (LWL) 81.7949%

```
 weka.classifiers.lazy.LWL -U 0 -K -1 -A
"weka.core.neighboursearch.BallTree -A \"weka.core.EuclideanDistance -R
first-last\" -C \"weka.core.neighboursearch.balltrees.TopDownConstructor
-S weka.core.neighboursearch.balltrees.PointsClosestToFurthestChildren
-N 40\"" -W weka.classifiers.bayes.NaiveBayesMultinomial
```

Ο RandomSubSpace που επί της ουσίας είναι ο Fast decision tree learner (REPTree) 73.8462%

```
weka.classifiers.meta.RandomSubSpace -P 0.5 -S 1 -num-slots 1 -I 10 -W
weka.classifiers.trees.REPTree -- -M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0
```

Ο RIPPER (Jrip) 70.8333%

```
weka.classifiers.rules.JRip -F 3 -N 2.0 -O 2 -S 1
```

Ο DecisionTable 54.0385%

```
weka.classifiers.rules.DecisionTable -X 1 -S
"weka.attributeSelection.BestFirst -D 1 -N 5"
```

Ο OneR 32.6923%

```
weka.classifiers.rules.OneR -B 6
```

Δυστυχώς, **δεν βρήκα αλγόριθμο με παραμέτρους που να πετυχαίνω ποσοστό σωτά ταξινομημένων στιγμιοτύπων μεγαλύτερο του 90%**. Έκανα και άλλα πειράματα με τους αλγορίθμους IBK, K-Star, PART, Hoeffding Tree, J48 κλπ. με διάφορες παραλλαγές και παραμέτρους και δεν βρήκα ικανοποιητικό αποτέλεσμα. Φαίνεται ότι τα data είναι πολλά και πολύ πιθανόν μη γραμμικά συσχετισμένα για να τα διαχειριστεί αποδοτικά κάποιος αλγόριθμος από αυτούς που δοκίμασα. Επίσης, δοκίμασα και από την εφαρμογή των πειραμάτων μου τον multilayer perceptron γιατί στο weka έσκαγε “out of memory exception” όμως για πολλές ώρες δεν έφερνε αποτέλεσμα. Δηλαδή δεν μπορούσε το μηχάνημα μου να διαχειριστεί αποδοτικά τον αλγόριθμο με αυτά τα δεδομένα. Υποψιάζομαι οτι ο multilayer perceptron θα έδινε ικανοποιητικό αποτέλεσμα για κατάλληλες παραμέτρους σε ένα δυνατότερο μηχάνημα ή αν εφαρμοστεί πρωτίστως ένας αλγόριθμος μείωσης των δεδομένων.

ΠΑΡΑΠΟΜΠΕΣ

- https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31
- https://researchcommons.waikato.ac.nz/handle/10289/5701
- https://en.wikipedia.org/wiki/Hinge_loss
- https://programmathically.com/understanding-hinge-loss-and-the-svm-cost-function/
- https://weka.sourceforge.io/doc.dev/weka/classifiers/functions/SGD.html
- https://www.futurelearn.com/info/courses/more-data-mining-with-weka/0/steps/29142
- https://machinelearningmastery.com/design-and-run-your-first-experiment-in-weka/
- https://github.com/renatopp/arff-datasets/blob/master/classification/mushroom.arff
- https://users.cs.utah.edu/~zhe/pdf/lec-19-2-svm-sgd-upload.pdf
- https://www.youtube.com/watch?v=Gw5s3yYVQAE&ab_channel=HowTo