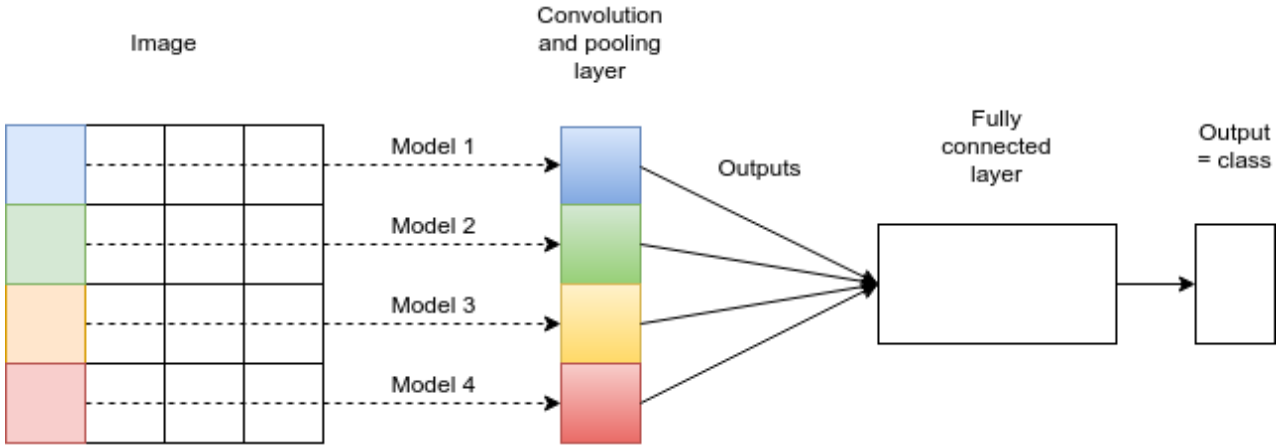
Κεφάλαιο 4

ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ

Συνδυασμός Πολλαπλών Μοντέλων

Combining Multiple Models (meta-algorithms/ ensemble techniques)

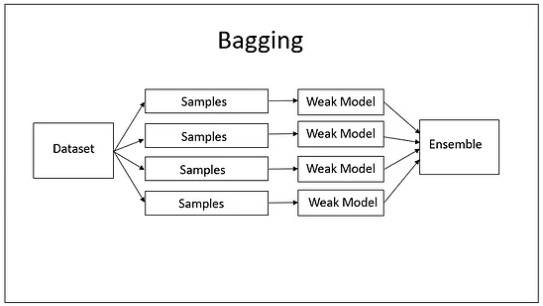


Εισαγωγή

- Ένα μοντέλο πρόβλεψης που προκύπτει από Μηχανική Μάθηση δεν είναι πάντα τέλειο.
  - Η απόδοση του κυμαίνεται ανάλογα με τον αριθμό και την ποιότητα των δεδομένων και την καταλληλότητα του αλγορίθμου που χρησιμοποιήθηκε σε σχέση με τα δεδομένα.
  - Πως θα μπορούσαμε να βελτιώσουμε την απόδοση του;
- Μια προφανής λύση είναι να συνδυάσουμε τις αποφάσεις πολλών διαφορετικών μοντέλων πρόβλεψης *(ensemble techniques)*. Οι πιο διαδεδομένοι τρόποι είναι:
  - Εφαρμόζοντας τον ίδιο αλγόριθμο σε διαφορετικά υποσύνολα δεδομένων
    - ✓ Μέθοδοι ενθυλάκωσης (*Bagging*) και ενίσχυσης (*Boosting*)
  - Εφαρμόζοντας τον ίδιο αλγόριθμο σε υποσύνολα χαρακτηριστικών
    - ✓ Αλγόριθμος τυχαίου δάσους (*random forest*)
  - Εφαρμόζοντας διαφορετικούς αλγορίθμους στο αρχικό σύνολο δεδομένων
    - ✓ Όπως η συσσώρευση ή στοίβαξη (*Stacking*), όπου οι προβλέψεις από μοντέλα προερχόμενα από έναν ή περισσότερους αλγόριθμους, συνδυάζονται σε ένα μετα-μοντέλο για να παραχθεί η τελική πρόβλεψη
  - Μετατρέποντας ένα πρόβλημα ταξινόμησης πολλών κλάσεων, σε πολλά προβλήματα δύο κλάσεων, όπως συμβαίνει στον αλγόριθμο *Error-Correcting Output Codes* (*ECOC*)
  - Εφαρμόζοντας επαναληπτικά κάποιον αλγόριθμο με διαφορετικές αρχικές παραμέτρους,
    - ✓ όπως για παράδειγμα ένα νευρωνικό δίκτυο που αλλάζει δομή ή αρχικές τιμές βαρών
- Οι παραπάνω μέθοδοι σχεδόν πάντα αυξάνουν την απόδοση πρόβλεψης.
  - Ωστόσο, το συνδυασμένο μοντέλο είναι πολύπλοκο και είναι δύσκολο να αναλυθούν οι παράγοντες που συνεισφέρουν στην τελική συνδυασμένη απόφαση (non explainable).

Bagging (Bootstrap Aggregating) -Ενθυλάκωση ή συνάθροιση αυτοδυναμίας

- Εφαρμόζεται σε αλγορίθμους μάθησης οι οποίοι είναι ασταθείς, δηλ. έχουν μεγάλη διακύμανση (variance)
  - Δηλαδή μικρές αλλαγές στα δεδομένα εκπαίδευσης προκαλούν αλλαγές και στο μοντέλο με αποτέλεσμα αυτό να βγάζει διαφορετικές αποφάσεις για κάποιες περιπτώσεις.
    - ✓ Τέτοιοι αλγόριθμοι είναι για παράδειγμα τα δένδρα απόφασης.
- Το Bagging αρχικά δημιουργεί (τυχαία) πολλά διαφορετικά σύνολα δεδομένων από το αρχικό μέσω "Δειγματοληψίας με Επανατοποθέτηση".
  - Το νέο σύνολο δεδομένων έχει ίδιο αριθμό δεδομένων με το αρχικό, αλλά κάποια δεδομένα έχουν επαναληφθεί ενώ κάποια δεν έχουν συμπεριληφθεί καθόλου.
  - Στη συνέχεια εφαρμόζει τον ασταθή αλγόριθμο μάθησης σε όλα τα νέα σύνολα δεδομένων και παράγει αντίστοιχα μοντέλα πρόβλεψης.

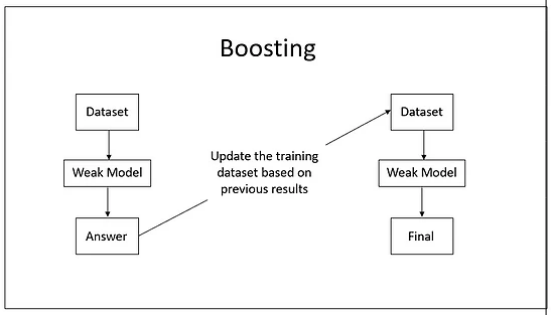


- Για τη διαδικασία πρόβλεψης λαμβάνουμε υπόψη τις αποφάσεις όλων των μοντέλων:
  - Η τελική τιμή είναι είτε η κλάση που συγκεντρώνει τις περισσότερες αποφάσεις μοντέλων (*voting*) είτε ο μέσος όρος των αριθμητικών προβλέψεων των διαφορετικών μοντέλων.
    - ✓ Ο αλγόριθμος που χρησιμοποιούμε ονομάζεται βασικός εκτιμητής (base estimator).
  - Αυτή η μέθοδος χρησιμοποιείται στον αλγόριθμο τυχαίου δάσους (*random forest*) ο οποίος συνδυάζει πολλά δένδρα απόφασης
    - ✓ Test: Weka, data set:titanic, compare J48 / Bagging with J48
  - Στο sklearn: sklearn.ensemble.BaggingClassifier και sklearn.ensemble.BaggingRegressor
    - ✓ estimator οποιοσδήποτε με default τα δένδρα (DecisionTreeClassifier ή DecisionTreeRegressor)

Β) Boosting: Εισαγωγή (1/3)

- Boosting (ενίσχυση) στη Μηχανική Μάθηση, είναι μια επαναληπτική διαδικασία που μετατρέπει αδύναμους αλγορίθμους σε ισχυρούς ελαττώνοντας το bias και το variance
  - ✓ Λεπτομέρειες για bias και variance στο Kef06-Model Evaluation
- Στηρίζεται στην ανάθεση βαρών (θετικών αριθμών) στα δεδομένα, έτσι ώστε ο αλγόριθμος μάθησης να επικεντρωθεί σε δεδομένα που συνήθως ταξινομούνται λάθος (classification).
  - ❑ Αν ο αλγόριθμος μπορεί να χειριστεί βάρη στα δεδομένα, τότε δεν υπάρχει πρόβλημα στην εφαρμογή του Boosting.
    - ✓ Για παράδειγμα ο C4.5 (και ο C5) μπορεί να χειριστεί βάρη υπολογίζοντας τη συνεισφορά ενός δεδομένου στην εντροπία ανάλογα με το βάρος του.
    - ✓ Παράμετρος sample_weight στο fit (Link)
  - ❑ Αν ο αλγόριθμος δεν μπορεί να χειριστεί βάρη, τότε μετατρέπουμε ένα σύνολο δεδομένων με βάρη σε ένα χωρίς, μέσω "Δειγματοληψίας με Επανατοποθέτηση"
    - ✓ Η πιθανότητα επιλογής ενός δεδομένου κατά τη δειγματοληψία είναι ανάλογη του βάρους του.
    - ✓ Έτσι δεδομένα με μεγαλύτερο βάρος εμφανίζονται περισσότερες φορές ενώ δεδομένα με μικρότερο βάρος μπορεί να μην εμφανιστούν καθόλου.
- Τα βάρη αλλάζουν τον τρόπο υπολογισμού της απόδοσης ενός αλγορίθμου:
  - ❑ Χωρίς βάρη, το ποσοστό λάθους είναι ο αριθμός των δεδομένων ελέγχου (test data) που ταξινομούνται λάθος προς το συνολικό αριθμό των δεδομένων ελέγχου.
  - ❑ Με βάρη, είναι το άθροισμα των βαρών των δεδομένων ελέγχου που ταξινομούνται λάθος προς το συνολικό άθροισμα των βαρών του συνόλου των δεδομένων ελέγχου (σταθμισμένο σφάλμα).

Boosting: Η διαδικασία (2/3)



- Το Boosting είναι μια επαναληπτική διαδικασία:
  - Τα διαφορετικά μοντέλα κατασκευάζονται το ένα μετά το άλλο. Η απόδοση του προηγούμενου μοντέλου επηρεάζει την κατασκευή του επόμενου.
    - ✓ Συγκεκριμένα προσπαθεί να κατασκευάσει το επόμενο μοντέλο έτσι ώστε να μην κάνει τα ίδια λάθη με αυτά που έκανε το προηγούμενο.
    - ✓ Αυτό επιτυγχάνεται χρησιμοποιώντας τα βάρη στα δεδομένα εκπαίδευσης.
- Διαδικασία Boosting (με μεταβολή βαρών):
  - 1. Το πρώτο μοντέλο παράγεται από το αρχικό σύνολο δεδομένων. Αν ο αλγόριθμος χειρίζεται βάρη τότε θέτουμε σε όλα τα δεδομένα του αρχικού συνόλου N ίσα βάρη (1/N).
  - 2. Έπειτα τα δεδομένα ταξινομούνται από το μοντέλο
    - ✓ Αν το σταθμισμένο σφάλμα e είναι πάνω από 50% (e > 0.5), το μοντέλο απορρίπτεται, τα βάρη τίθενται στην τιμή 1/N και η διαδικασία επιστρέφει στο προηγούμενο βήμα
  - 3. Αν η απόφαση του μοντέλου για κάποιο δεδομένο είναι λάθος τότε το βάρος του αυξάνεται ενώ αν είναι σωστή μειώνεται
  - 4. Η διαδικασία επαναλαμβάνεται από το βήμα 2 για τη μάθηση του επόμενου μοντέλου έως ότου επιτευχθεί ένα επιθυμητό όριο σφάλματος ή για προκαθορισμένο πλήθος κύκλων ενίσχυσης
    - ✓ Υπενθυμίζεται ότι αν ο αλγόριθμος δεν χειρίζεται βάρη, θα πρέπει το σύνολο δεδομένων εκπαίδευσης να προκύψει από το προηγούμενο με "Δειγματοληψία με Επανατοποθέτηση".
- Με το Boosting, παράγονται επαναληπτικά μοντέλα πρόβλεψης που επικεντρώνονται στα δύσκολα δεδομένα τα οποία δεν ταξινομούνται σωστά από το αρχικό μοντέλο

Boosting: Πόσο αλλάζουν τα βάρη; (3/3)

- Πόσο πρέπει να αλλάζουν τα βάρη των δεδομένων μετά από κάθε κύκλο μάθησης;
  - Για δεδομένα που ταξινομούνται λάθος το βάρος παραμένει όσο ήταν αρχικά, ενώ για αυτά που ταξινομούνται σωστά το βάρος μειώνεται αντιστρόφως ανάλογα με το ποσοστό λαθών e του ταξινομητή στα δεδομένα:

$$weight = weight\,\frac{e}{1-e}$$

  - Στη συνέχεια τα βάρη κανονικοποιούνται έτσι ώστε το άθροισμα τους να παραμείνει όσο και πριν.
    - ✓ Κάθε βάρος διαιρείται με το άθροισμα των νέων βαρών και πολλαπλασιάζεται με το άθροισμα των παλιών.
    - ✓ Έτσι αυτόματα αυξάνεται το βάρος των δεδομένων που ταξινομούνται λάθος και μειώνεται αυτών που ταξινομούνται σωστά.
- Η διαδικασία επαναλαμβάνεται μέχρι το ποσοστό λάθους του τρέχοντος μοντέλου γίνει είτε 0 είτε μεγαλύτερο ή ίσο του 0.5.
  - Στη δεύτερη περίπτωση, το τελευταίο αυτό μοντέλο διαγράφεται.
- Για την ταξινόμηση άγνωστων δεδομένων, συνδυάζονται οι αποφάσεις όλων των μοντέλων μέσω ψηφοφορίας με βάρη.
  - Το βάρος της απόφασης κάθε μοντέλου είναι αντίστοιχο του ποσοστού λαθών e στα δεδομένα από τα οποία εκπαιδεύτηκε:

$$weight = -\,log\,\frac{e}{1-e}$$

Boosting στο Regression

- Ισχύουν τα ίδια με τα προηγούμενα με τη διαφορά ότι υπολογίζει τα απόλυτο, τετραγωνικό ή εκθετικό σφάλμα (e) για κάθε παράδειγμα και τα κανονικοποιεί [0,1] διαιρώντας με το μέγιστο.
  - Το βάρος (w) ενός παραδείγματος ενημερώνεται βάσει του τύπου:
  - $w' = w * \beta^{(1-e)*\lambda}$ $όπου\ \beta = \frac{\sum e}{1-\sum e}$ και λ είναι το learning rate
    - ✓ scikit-learn/sklearn/ensemble/ _weight_boosting.py
- Python (sklearn) ----------
  - AdaBoost classifier
    - ✓ Python (sklearn): sklearn.ensemble.AdaBoostClassifier (*base_estimator*, *n_estimators, learning_rate, algorithm={SAMME, SAMME.R})*
  - AdaBoost Regressor
    - ✓ Python (sklearn): sklearn.ensemble.AdaBoostRegressor (*estimator*, *n_estimators, learning_rate, loss={linear, square, explonential})*
  - SAMME (Stagewise Additive Modeling using a Multi-class Exponential loss)
    - ✓ is an algorithm used in multiclass classification for boosting weak learners (typically decision trees) into a strong ensemble classifier.
    - ✓ SAMME is an extension of the AdaBoost (Adaptive Boosting) algorithm and is designed to handle problems with more than two classes.
  - SAMME.R (Stagewise Additive Modeling using a Multi-class Exponential loss with Real-valued Predictions) is an enhancement of the original SAMME algorithm designed for multiclass classification.

Συμπέρασμα

- Το boosting είναι ένας μετα-αλγόριθμος που συνδυάζει μοντέλα μηχανικής μάθησης με σκοπό κυρίως τη μείωση της μεροληψίας αλλά και της διακύμανσης στην επιβλεπόμενη μάθηση.
  - Προτάθηκε στο πλαίσιο του αλγορίθμου Adaboost (*adaptive boosting*) ο οποίος χρησιμοποιεί δένδρα ταξινόμησης/παρεμβολής σαν βασικό αλγόριθμο
  - Το boosting μετατρέπει αδύναμους αλγορίθμους σε ισχυρούς
    - ✓ Είτε μεταβάλλοντας τα βάρη του συνόλου εκπαίδευσης (Adaboost)
    - ✓ Είτε μέσω μιας διαδικασία ελαχιστοποίησης μιας κυρτής συνάρτησης κόστους
- Η *διαβαθμισμένη ενίσχυση* (*Gradient boosting*) είναι μια τεχνική που αντί να παράγει μοντέλα μεταβάλλοντας τα βάρη του συνόλου εκπαίδευσης, ενισχύει ένα αδύναμο αλγόριθμο μέσω μιας διαδικασίας βελτιστοποίησης διαβαθμισμένης καθόδου (*gradient descent optimization procedure*) ελαχιστοποιώντας μια κατάλληλη συνάρτηση κόστους.
  - Αλγόριθμοι βασισμένοι στο boosting πετυχαίνουν καλύτερα αποτελέσματα από τον Adaboost, όπως οι XGBoost, LPBoost, Totalboost, BrownBoost, LogitBoost, MadaBoost και άλλοι.
- Ο *XGBoost* (*Extreme Gradient Boosting*) είναι ένας αλγόριθμος ενίσχυσης ενός δένδρου (ταξινόμησης/παρεμβολής) μέσω της *διαβαθμισμένης ενίσχυσης* (*gradient boosting*).
  - Προτάθηκε το 2016 από τους Tianqi Chen και Carlos Guestrin και απέκτησε μεγάλη δημοσιότητα καθώς χρησιμοποιήθηκε από πολλές ομάδες που διακρίθηκαν σε διαγωνισμούς μηχανικής μάθησης.

https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/

Gradient Boosting (Διαβαθμισμένη ενίσχυση)

- Is a machine learning technique for regression and classification problems, which produces a model in the form of an ensemble of weak prediction models, typically decision trees.
  - It builds the models in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.
- Gradient boosting involves three main steps.
  - The first step that is required is that a loss function be optimized.
  - The loss function must be differentiable.
    - ✓ A loss function measures how well a machine learning model fits the data of a certain phenomenon. It depends on the type of problem.
    - ✓ For example, regression may use a squared error and classification may use logarithmic loss.
  - The second step is the use of a weak learner.
    - ✓ In gradient boosters, the weak learner is a decision tree.
    - ✓ Specifically regression trees are used that output real values for splits and whose output can be added together, allowing subsequent models outputs to be added to correct the residuals in the predictions of the previous iteration.
  - The third step is combing many weak learners in an additive fashion.
    - ✓ Decision trees are added one at a time. A gradient descent procedure is used to minimize the loss when adding trees.
    - ✓ That's the gradient part of gradient boosters.

Gradient descent vs. Gradient boosting

- Gradient descent optimization in the machine learning world is typically used to find the parameters associated with a single model that optimizes some loss function.
  - ✓ The gradient descent optimization occurs on the output of the model.
- In contrast, gradient boosters are meta-models consisting of multiple weak models whose output is added together to get an overall prediction.
- In the figure we can see that gradient boosting adds sub-models incrementally to minimize a loss function.
- The boosting ensemble technique consists of three simple steps:
  - ✓ An initial model F_0 is defined to predict the target variable y. This model will be associated with a residual ($y – F_0$)
  - ✓ A new model h_1 is fit to the residuals from the previous step
  - ✓ Now, F_0 and h_1 are combined to give F_1, the boosted version of F_0. The mean squared error from F_1 will be lower than that from F_0:

$$F_1(x) <- F_0(x) + h_1(x)$$

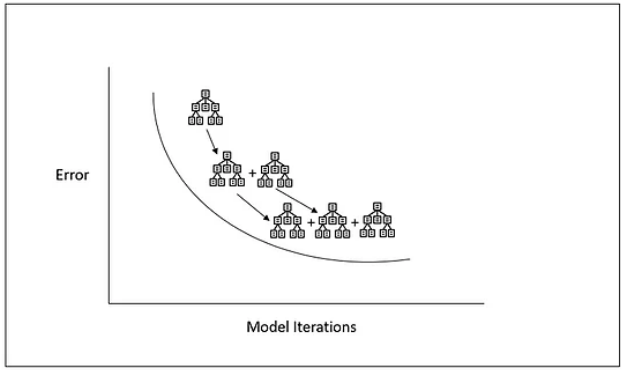
    - Δηλ. Η πρόβλεψη για το x του F_1 είναι το άθροισμα των προβλέψεων του $F_0(x)$ και $h_1(x)$ και έχει μικρότερο σφάλμα
  - ✓ To improve the performance of F_1, we could model after the residuals of F_1 and create a new model F_2:

$$F_2(x) <- F_1(x) + h_2(x)$$

  - ✓ This can be done for ‘m’ iterations, until residuals have been minimized as much as possible:

$$F_m(x) <- F_{m-1}(x) + h_m(x)$$

  - ✓ Here, the additive learners do not disturb the functions created in the previous steps. Instead, they impart information of their own to bring down the errors. [Introduction to XGBoost Algorithm in Machine Learning]



Boosting as an optimization algorithm

- Boosting can be seen as minimization of a convex loss function over a convex set of functions.
  - Specifically, the loss being minimized by AdaBoost is the exponential loss

$$\sum_i \phi(i, y, f) = \sum_i e^{-y_i f(x_i)},$$

  - whereas LogitBoost performs logistic regression (i.e. classification), minimizing Log loss

$$\sum_i \phi(i, y, f) = \sum_i \ln\left(1 + e^{-y_i f(x_i)}\right)$$

  - A Unified View of Loss Functions in Supervised Learning

- Gradient Boosting builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.
- The idea of Gradient Boosting originated in the observation by Leo Breiman that boosting can be interpreted as an optimization algorithm on a suitable cost function.

AdaBoost vs. GBoost vs. XGBoost

- Οι διαφορές που εντοπίζονται στους αλγορίθμους AdaBoost, GBoost και XGBoost, σχετίζονται με τη συνάρτηση κόστους που χρησιμοποιεί ο καθένας από αυτούς
- Παρατηρήθηκε θεωρητικά ότι ο AdaBoost μπορεί να γενικευτεί σε έναν αλγόριθμο (*Gradient Boost - GBoost*) που θα μπορεί να χρησιμοποιεί διαφορετικές συναρτήσεις κόστους για τη βελτιστοποίηση των μοντέλων που συνδυάζονται
  - ή το αντίστροφο, δηλαδή ο AdaBoost είναι ένας gradient boost αλγόριθμος που χρησιμοποιεί μια εκθετική (exponential) συνάρτηση κόστους
    - ✓ Η συνάρτηση κόστους του AdaBoost είναι αυτή που διορθώνει τα επιμέρους σφάλματα του κάθε μοντέλου (με προσθετικό τρόπο)
  - Ο GBoost ελαχιστοποιεί οποιαδήποτε συνάρτηση κόστους μέσω *gradient descent*
    - ✓ Το μειονέκτημα είναι ότι στηρίζεται σε πολλές υπερ-παραμέτρους
  - Ο XGBoost στηρίζεται στον GBoost, καθώς η λειτουργικότητά του είναι παρόμοια, αλλά με σημαντικές βελτιώσεις
    - ✓ Κύρια επέκταση είναι η εφαρμογή μεθόδου ομαλοποίησης (regularization)
    - ✓ Άλλες επεκτάσεις είναι υποστήριξη παράλληλης επεξεργασίας, διαχείριση ελλιπών τιμών, κλάδεμα δέντρου, ενσωματωμένη μέθοδο cross-validation, κ.α.
    - ✓ What is the difference between eXtreme Gradient Boosting (XGBoost), AdaBoost, and Gradient Boosting?
    - ✓ Regularization in Machine Learning

Regularization (Ομαλοποίηση)

- A technique that discourages learning a more complex model, so as to avoid the risk of overfitting.
- A term called *regularizer* is added to the cost function: $Cost(f) = Loss(f) + \lambda \cdot Complexity(f)$
  - The regularizer is a penalty added to the cost function that shrinks model parameters towards the zero vector using either the squared euclidean norm L2 or the absolute norm L1 or a combination of both (Elastic Net).
    - ✓ λ is the tuning parameter that decides how much we want to penalize the complexity of our model.
    - ✓ Selecting (with Cross validation?) a good value of λ is critical.
- The *Complexity(f)* should have such a form that it weights the cost function when the weights take large values.
- A simple form of *Complexity(f)* is: $\Sigma|w_i|^n$ where
  - For n=1 we obtain the L1 norm regularization or lasso) and
  - For n=2 we obtain the L2 norm regularization or ridge).
  - Elastic Net (L1+L2 norm regularization)
- Why do we want to contain two component in the Cost (or objective) function?
  - Optimizing training loss encourages predictive models fitting well in training data
  - Optimizing regularization encourages simple models
    - ✓ Simpler models tends to have smaller variance in future predictions, making prediction stable
    - ✓ Python command (sklearn): sklearn.linear_model.Lasso(alpha=1.0)
    - ✓ Regularization in Machine Learning

- Example
  - A simple relation for linear regression looks like this: $Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p$
  - The fitting procedure involves a loss function, like Residual Sum of Squares or RSS.
  - The coefficients are chosen, such that they minimize this loss function

$$\mathrm{RSS} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2$$

- Ridge Regression or L2 norm

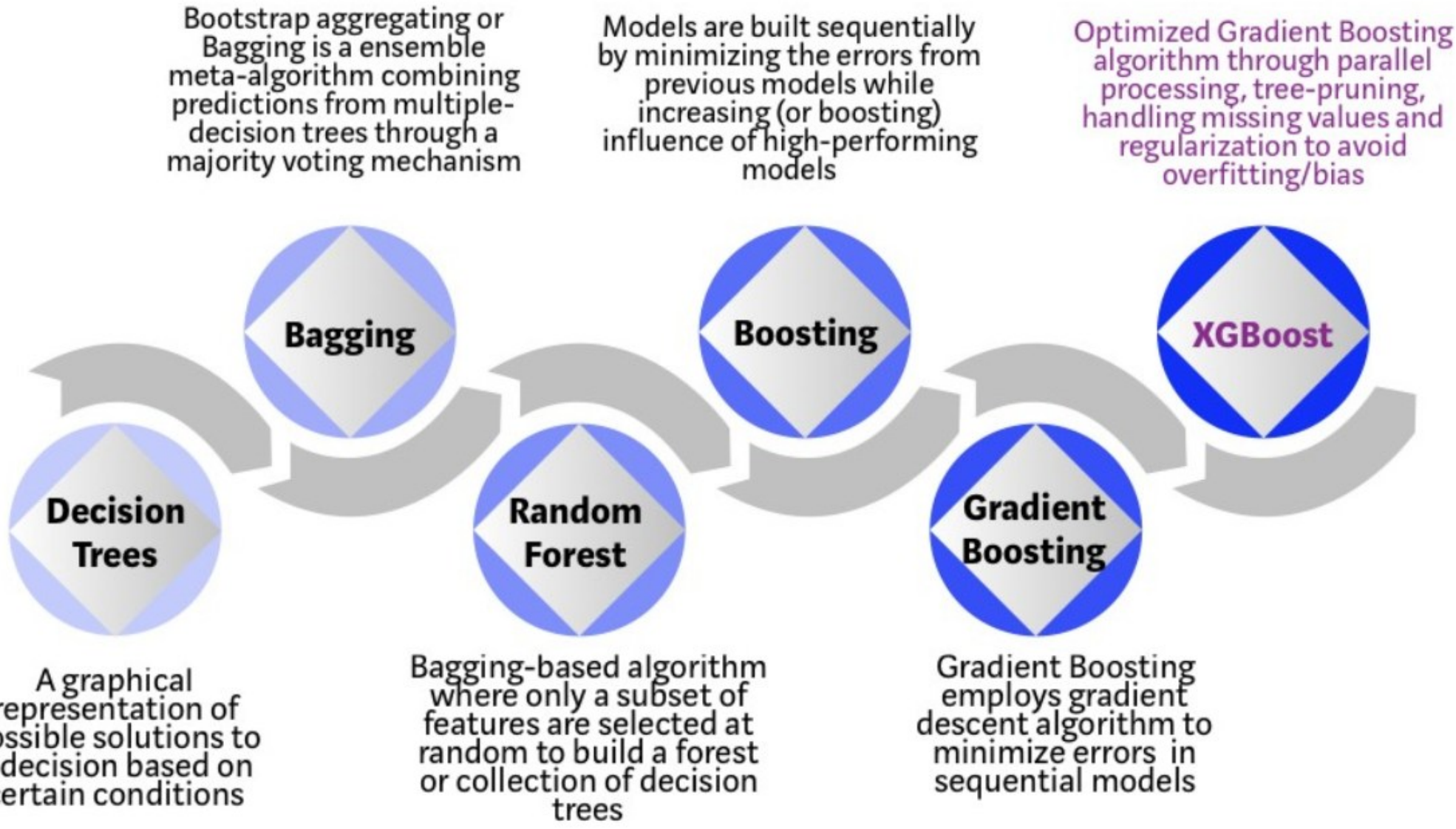
Cost(f)= $$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 = \mathrm{RSS} + \lambda \sum_{j=1}^{p} \beta_j^2$$

  - RSS is modified by adding a shrinkage quantity and the coefficients are estimated by minimizing this function.
- Lasso L1 norm is another variation, in which the following function is minimized

Cost(f)= $$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| = \mathrm{RSS} + \lambda \sum_{j=1}^{p} |\beta_j|.$$

- These techniques are <u>penalizing the high coefficients.</u>

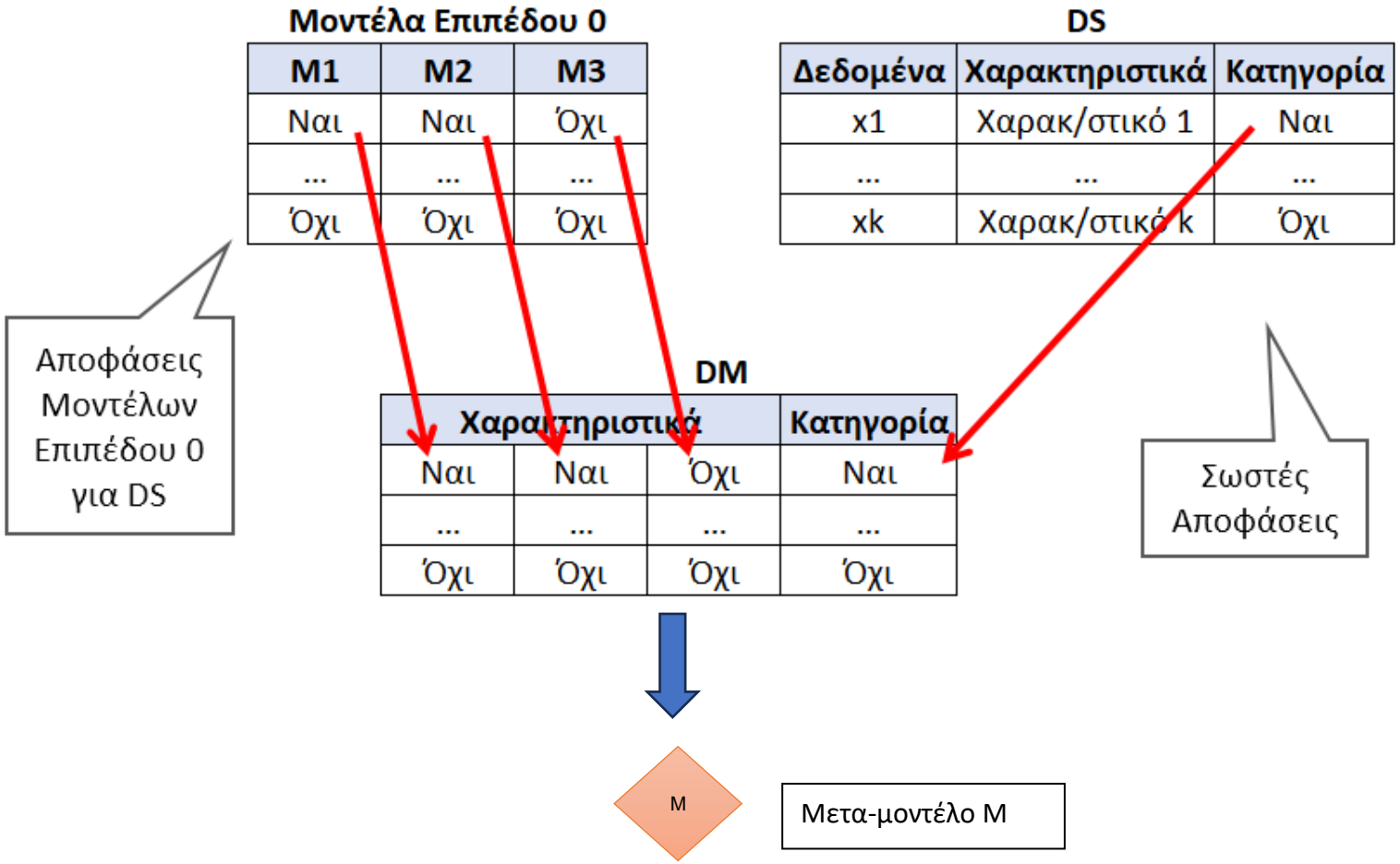
Tree-based algorithms evolution



Γ) Stacking (1/3)

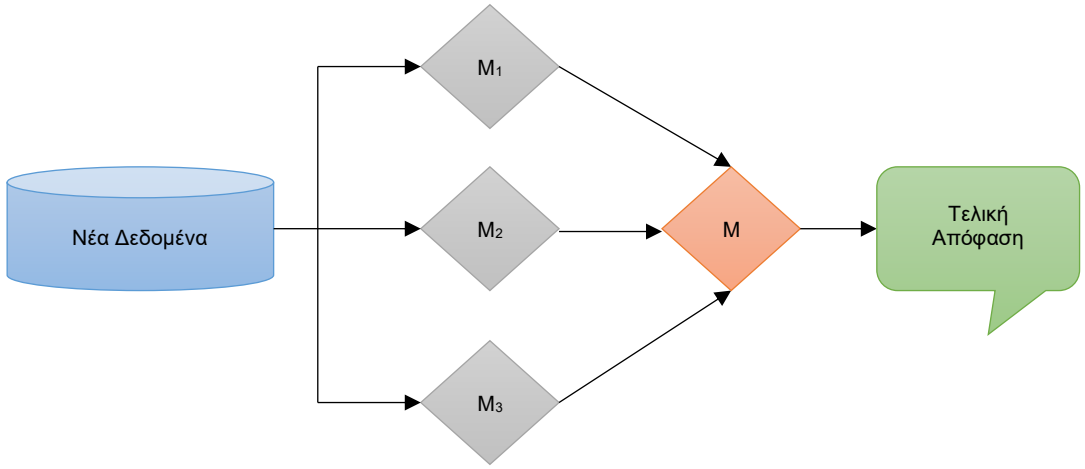
- Σε αντίθεση με τις μεθόδους Bagging και Boosting, η μέθοδος Stacking χρησιμοποιείται συνήθως για τον συνδυασμό μοντέλων πρόβλεψης που προέκυψαν από διαφορετικούς αλγορίθμους μάθησης.
- Το Stacking στηρίζεται στην παραγωγή ενός μετά-μοντέλου (μοντέλο επιπέδου 1) με δεδομένα εκπαίδευσης τις αποφάσεις ενός συνόλου μοντέλων (μοντέλα επιπέδου 0).
  - Το μετά-μοντέλο αυτό προσπαθεί να μάθει ποια μοντέλα είναι τα πιο αξιόπιστα από αυτά που συμμετέχουν, έτσι ώστε να δίνει την σωστή απόφαση κάθε φορά.
- Παράδειγμα: Ας υποθέσουμε ότι έχουμε ένα σύνολο δεδομένων D.
  - Αρχικά το χωρίζουμε σε δύο σύνολα: *DT* (δεδομένα εκπαίδευσης) και *DS* (δεδομένα Stacking)
    - ✓ Συνήθως το DS αποτελεί το 10% του D.
  - Έστω ότι παράγουμε (εκπαιδεύουμε) τρία μοντέλα πρόβλεψης *M*1, *M*2, *M*3 (μοντέλα επιπέδου 0), εφαρμόζοντας για παράδειγμα τους αλγορίθμους Naive Bayes, kNN και C4.5 αντίστοιχα, στο σύνολο δεδομένων εκπαίδευσης *DT*.
  - Στη συνέχεια, για κάθε δεδομένο stacking του συνόλου *DS* λαμβάνουμε τις αποφάσεις των μοντέλων *M1*, *M2* και *M3* και μαζί με την σωστή απόφαση που γνωρίζουμε από το *DS* σχηματίζουμε ένα δεδομένο του συνόλου μετά-εκπαίδευσης *DM*.
  - Από το σύνολο μετά-εκπαίδευσης *DM*, χρησιμοποιώντας κάποιον αλγόριθμο μάθησης της προτίμησης μας, παράγουμε το τελικό μετά-μοντέλο *M*.

Stacking: Παραγωγή Μετα-δεδομένων (2/3)



Stacking: Ταξινόμηση νέων δεδομένων (3/3)

- Για την ταξινόμηση ενός νέου δεδομένου, πρώτα παίρνουμε τις απαντήσεις των μοντέλων επιπέδου 0 (M1, M2 και M3) για αυτό το δεδομένο και στη συνέχεια τις τροφοδοτούμε στο μετά-μοντέλο M, το οποίο και μας δίνει μια τελική απάντηση.



- Η απόδοση του stacking αναμένεται να είναι καλύτερη από τις αποδόσεις των επιμέρους μοντέλων που χρησιμοποιούνται.
- Έχει χρησιμοποιηθεί με επιτυχία τόσο σε επιβλεπόμενη μάθηση (ταξινόμηση και παρεμβολή) όσο και σε μη επιβλεπόμενη μάθηση (εκτίμηση πυκνότητας).
  - Εντούτοις χρησιμοποιείται λιγότερο σε σχέση με τις μεθόδους bagging και boosting.
    - ✓ Introduction to Ensembling/Stacking in Python

Ensemble classification methods (Python)

- Bagging
  - Python (sklearn): sklearn.ensemble.BaggingClassifier (*base_estimator, n_estimators, max_samples, bootstrap={True, False}, bootstrap_features={True, False}, oob_score={True, False}*)
- AdaBoost classifier
  - Python (sklearn): sklearn.ensemble.AdaBoostClassifier (*base_estimator*, *n_estimators, learning_rate, algorithm={SAMME, SAMME.R})*
- Gboost ensemble classifier
  - Python (sklearn): sklearn.ensemble.GradientBoostingClassifier (*loss={deviance, exponential}, learning_rate, n_estimators, subsample, criterion={MAE, MSE, Friedman MSE}, min_samples_split, min_samples_leaf, max_depth, max_features*)
    - ✓ Το n_estimators ρυθμίζει μόνο πόσα base models να εκπαιδεύσει, αλλά όχι με ποιόν αλγόριθμο
    - ✓ Έχει συγκεκριμένο base learner. Θα πρέπει να γίνει custom λύση για να αλλάξει
- XGBoost classifier
  - Python (XGBoost): xgboost.XGBClassifier (*eta (learning rate), booster = {dart, gblinear, gbtree}, objective={reg:logistic, binary:logistic, multi:softmax, …}, eval_metric={RMSE, MAE, logloss, error, …}*)
- Stacking classifier
  - Python (sklearn): sklearn.ensemble.StackingClassifier (*estimators, final_estimator, cv (cross-validation splitting strategy)*)

Ensemble regression methods (Python)

- Bagging Regressor
  - Python (sklearn): sklearn.ensemble.BaggingRegressor (*estimator, n_estimators, max_samples, bootstrap={True, False}, bootstrap_features={True, False}, oob_score={True, False}*)
- AdaBoost Regressor
  - Python (sklearn): sklearn.ensemble.AdaBoostRegressor (*estimator, n_estimators, learning_rate, loss={linear, square, explonential})*
- Gboost ensemble Regressor
  - Python (sklearn): sklearn.ensemble.GradientBoostingRegressor (*loss={squared_error, absolute_error, huber}, learning_rate, n_estimators, subsample, criterion={MAE, MSE, Friedman MSE}, min_samples_split, min_samples_leaf, max_depth, max_features*)
    - ✓ Το n_estimators ρυθμίζει μόνο πόσα base models να εκπαιδεύσει, αλλά όχι με ποιόν αλγόριθμο
    - ✓ Έχει συγκεκριμένο estimator. Θα πρέπει να γίνει custom λύση για να αλλάξει.
- XGBoost Regressor
  - Python (XGBoost): xgboost.XGBRegressor (*eta (learning rate), booster = {dart, gblinear, gbtree}, objective={reg:logistic, binary:logistic, multi:softmax, …}, eval_metric={RMSE, MAE, logloss, error, …}*)
- Stacking Regressor
  - Python (sklearn): sklearn.ensemble.StackingRegressor (*estimators, final_estimator, cv (cross-validation splitting strategy)*)

Libraries

- LightGBM
  - Light Gradient-Boosting Machine, is a free and open-source distributed gradient-boosting framework for machine learning, originally developed by Microsoft.
  - It is based on decision tree algorithms and used for ranking, classification and other machine learning tasks.
  - The development focus is on performance and scalability.
    - ✓ Faster training speed and higher efficiency.
    - ✓ Lower memory usage.
    - ✓ Better accuracy.
    - ✓ Support of parallel, distributed, and GPU learning.
    - ✓ Capable of handling large-scale data.
    - ✓ (https://en.wikipedia.org/wiki/LightGBM)
- CatBoost
  - A high-performance open source library for gradient boosting on decision trees
  - Reduce overfitting when constructing models with a novel gradient-boosting scheme
    - ✓ Fast and scalable GPU version

Δ) Error-Correcting Output Codes (ECOC) (1/3)

- Για προβλήματα όπου υπάρχουν περισσότερες από 2 κατηγορίες/κλάσεις (*multiclass problem*), υπάρχει πολλές φορές η ανάγκη μετατροπής τους σε πολλά προβλήματα δύο κλάσεων ώστε να καταστούν επιλύσιμα από αλγορίθμους που χειρίζονται μόνο δύο κλάσεις (π.χ. SVM).
  - Η ίδια ανάγκη μπορεί να υπάρχει και σε αλγορίθμους που χειρίζονται προβλήματα πολλών κλάσεων, γιατί έτσι αυξάνεται η ακρίβεια πρόβλεψης τους.
- Υπάρχουν δύο τεχνικές μετατροπής:
  - Ένας εναντίον ενός (*one-against-one*) όπου αναπτύσσονται δυαδικοί ταξινομητές για κάθε δυνατό ζευγάρι τιμών κλάσης.
    - ✓ Για προβλήματα με k κλάσεις αναπτύσσονται συνολικά $k \cdot (k-1)/2$ ταξινομητές.
    - ✓ Για την τελική ταξινόμηση λαμβάνονται οι αποφάσεις από όλους τους ταξινομητές και χρησιμοποιούνται σχήματα ψηφοφορίας (voting) για την τελική απόφαση
    - ✓ Python (sklearn): sklearn.multiclass.OneVsOneClassifier(estimator)
  - Ένας εναντίον όλων (*one-against-all* ή *one-against-the-rest*) όπου κατασκευάζονται k σύνολα δεδομένων εκπαίδευσης και ένας δυαδικός ταξινομητής C_i για το καθένα από αυτά, ο οποίος προβλέπει τιμή +1 για την κλάση c_i και τιμή 0 για όλες τις υπόλοιπες
    - ✓ Python (sklearn): sklearn.multiclass.OneVsRestClassifier(estimator)

Παράδειγμα

- Έστω ένα σύνολο δεδομένων *D* με 4 κατηγορίες (κλάσεις) *α*, *β*, *γ*, *δ*
  - Για να εφαρμόσουμε τη μέθοδο *ένας-εναντίον-όλων*, μετατρέπουμε αυτό το σύνολο σε 4 σύνολα D_i με 2 μόνο κατηγορίες.
  - Κάθε σύνολο δημιουργείται ως αντίγραφο του αρχικού αλλά με αλλαγμένη τιμή για την κατηγορία.
  - Συγκεκριμένα στο πρώτο σύνολο κωδικοποιούμε την κατηγορία *α* ως 1 και τις *β*, *γ*, *δ* ως 0.
    - ✓ Στήλη D_1 στο σχήμα
  - Στο δεύτερο σύνολο κωδικοποιούμε την κατηγορία *β* ως 1 και τις *α*, *γ*, *δ* ως 0. (Στήλη D_2 στο σχήμα)
    - ✓ Ομοίως συνεχίζουμε και με τα υπόλοιπα σύνολα δεδομένων.
  - Εκπαιδεύουμε έναν ταξινομητή C_i για κάθε σύνολο.
  - Όταν έρθει ένα νέο δεδομένο το ταξινομούν και τα 4 μοντέλα.
  - Επιλέγουμε την κατηγορία για την οποία ο αντίστοιχος ταξινομητής δίνει έξοδο 1.
  - Φυσικά υπάρχει πρόβλημα αν δώσουν έξοδο 1 παραπάνω από ένας ταξινομητές.
  - Τη λύση δίνει η μέθοδος ECOC.

	4 Σύνολα Δεδομένων			
Κατηγορία	**D_1**	**D_2**	**D_3**	**D_4**
α	1	0	0	0
β	0	1	0	0
γ	0	0	1	0
δ	0	0	0	1
Ταξινομητής:	**C_1**	**C_2**	**C_3**	**C_4**

ECOC: Παράδειγμα (2/3)

- Η μέθοδος Κωδικοποίησης Διόρθωσης Σφαλμάτων Εξόδου (*Error-Correcting Output Codes* - *ECOC*) αντιμετωπίζει το παραπάνω πρόβλημα
  - ECOC is an ensemble method which combines many binary classifiers in order to solve a multi-class classification problem.
  - θεωρώντας την πρόβλεψη κάθε ταξινομητή C_i ως ένα bit και υιοθετώντας μια κωδικοποίηση περισσότερων ταξινομητών (περισσότερων bit),
    - ✓ Η τεχνική είναι εμπνευσμένη από τη διόρθωση λαθών σε κανάλια μετάδοσης ψηφιακού σήματος με θόρυβο.
- Έστω ότι στο προηγούμενο παράδειγμα χρησιμοποιούμε 7 σύνολα εκπαίδευσης αντί για 4 και συνεπώς 7 ταξινομητές (βλ. Σχήμα).
- Μπορούμε έτσι να κωδικοποιήσουμε κάθε κατηγορία από τις 4 με μία αλληλουχία 7-bit (αντί για 4 πριν) που διαφέρει από τις υπόλοιπες 7-δες αρκετά,
  - ώστε σε τυχόν λάθος πρόβλεψη κάποιου ταξινομητή C_i, να υπάρχει η δυνατότητα εντοπισμού του σφάλματος ή και διόρθωσης του τελικού αποτελέσματος της ομάδας.
  - Κάθε τέτοια σειρά bit αποτελεί τον κωδικό ECOC κάθε μιας κατηγορίας.

	7 Σύνολα Δεδομένων						
Κατηγορία	**D_1**	**D_2**	**D_3**	**D_4**	**D_5**	**D_6**	**D_7**
α	1	1	1	1	1	1	1
β	0	0	0	0	1	1	1
γ	0	0	1	1	0	0	1
δ	0	1	0	1	0	1	0
Ταξινομητής:	**C_1**	**C_2**	**C_3**	**C_4**	**C_5**	**C_6**	**C_7**

- Το πλεονέκτημα τώρα είναι ότι ακόμα και αν κάποιο μοντέλο κάνει λάθος, μπορούμε να βρούμε την πιο πιθανή κατηγορία εξετάζοντας την απόσταση του "κωδικού" που βγάζουν τα μοντέλα με τον "κωδικό" της κατηγορίας.
  - θα είναι αυτή με τη μικρότερη απόσταση Hamming από την έξοδο της ομάδας

ECOC: Επιλογή κατάλληλου κώδικα

- Διαχωρισμός Γραμμών (row separation)
  - Θα πρέπει η απόσταση δυο γραμμών, δηλαδή δύο κωδικών που ανατίθενται σε δύο διαφορετικές κατηγορίες να απέχουν όσο το δυνατόν περισσότερα bit, ώστε να μπορούν να διορθώνονται πιθανά λάθη των ταξινομητών.
    - ✓ Στον 1ο πίνακα η ελάχιστη απόσταση (d) είναι 2 (δηλαδή δύο bit πρέπει να αλλάξουν) και επομένως δεν μπορούμε να διορθώσουμε καθόλου λάθη, αφού ο Hamming κώδικας διορθώνει (d-1)/2 λάθη.
    - ✓ Στον 2ο πίνακα η ελάχιστη Hamming απόσταση είναι 4, οπότε μπορούν να διορθωθούν τα λάθη ενός μοντέλου.
  - Επομένως ένας καλός ECOC κώδικας πρέπει να πετυχαίνει το διαχωρισμό των γραμμών, ώστε να διορθώνει τα λάθη
  - Με πλήθος ταξινομητών $2^{\kappa-1}-1$ έχουμε σίγουρη διόρθωση ενός λάθους σε *κ* κλάσεις
    - ✓ Π.χ. για k=4 θέλουμε 2^3-1=7 ταξινομητές για τη διόρθωση ενός λάθους
- Διαχωρισμός στηλών (column separation)
  - Ένας καλός κώδικας ECOC θα πρέπει να επιτυγχάνει και διαχωρισμό στηλών, δηλαδή να απέχουν αρκετά bit και οι στήλες, δηλαδή οι κωδικοποιήσεις των κατηγοριών που κάνουν 2 διαφορετικά μοντέλα.
  - Αυτό είναι απαραίτητο γιατί αλλιώς, όποτε κάνει λάθος ένα μοντέλο θα κάνει και αυτό με την ίδια κωδικοποίηση. Δηλαδή θα είναι σαν να έχουμε το ίδιο μοντέλο 2 φορές.
- Με κάτω από 4 κατηγορίες δύσκολα καθορίζεται ένας καλός κώδικας ECOC.

  - Python (sklearn): sklearn.multiclass.OutputCodeClassifier(estimator, code_size)

Άλλες Τεχνικές Μάθησης με Επίβλεψη

- Εκτός από τις τεχνικές μηχανικής μάθησης με επίβλεψη που παρουσιάστηκαν στις προηγούμενες ενότητες, υπάρχουν και άλλες από τις οποίες οι σημαντικότερες είναι:
  - Πρόβλεψη Πολλαπλών Μεταβλητών
  - Ημι-επιβλεπόμενη Μάθηση (semi-supervised learning)
  - Ενεργή μάθηση (Active Learning)
  - Εξ'αποστάσεως επιβλεπόμενη μάθηση (distant supervised learning)
  - Learning to Rank

Πρόβλεψη Πολλαπλών Μεταβλητών

- Στη μάθηση με επίβλεψη η είσοδος είναι ένα ν-διάστατο διάνυσμα μεταβλητών εισόδου ενώ η έξοδος αποτελείται από μία μόνο μεταβλητή (εξαρτημένη μεταβλητή) η οποία μπορεί να είναι
  - ✓ είτε ονομαστική (ταξινόμηση) η οποία μπορεί να παίρνει 2 (binary) ή περισσότερες τιμές (multiclass), ή
  - ✓ συνεχής (παρεμβολή/παλινδρόμηση)
  - ✓ Η είσοδος αντιστοιχίζεται με μια κλάση ή με μια τιμή
- Ωστόσο, πολλά πραγματικά προβλήματα περιλαμβάνουν την αντιστοίχιση μίας εισόδου σε πολλές εξόδους ταυτόχρονα.
  - ✓ Για παράδειγμα, ένα άρθρο εφημερίδας μπορεί να ανήκει σε πολλές κατηγορίες (π.χ. πολιτική και τέχνη), μία εικόνα να απεικονίζει πολλά αντικείμενα και ένα μουσικό κομμάτι να περιλαμβάνει πολλά μουσικά όργανα ή να σχετίζεται με πολλά είδη μουσικής.
  - ✓ Σε τέτοιου είδους προβλήματα, η έξοδος αναπαρίσταται και αυτή με ένα διάνυσμα μεταβλητών των οποίων τις τιμές θέλουμε να προβλέψουμε ταυτόχρονα συναρτήσει των εισόδων.
- Ονομάζονται προβλήματα πρόβλεψης πολλαπλών μεταβλητών και περιλαμβάνουν προβλήματα με πολλαπλές ονομαστικές μεταβλητές εξόδου (*προβλήματα ταξινόμησης πολλαπλών ετικετών* - *multi-label classification*) και προβλήματα με συνεχείς μεταβλητές εξόδου (*προβλήματα παρεμβολής πολλαπλών στόχων* - *multi-target* ή *multi-output regression*).
  - ✓ Υπάρχει ένας σημαντικός αριθμός εξειδικευμένων αλγορίθμων, κυρίως για την ταξινόμηση πολλαπλών ετικετών (π.χ. η βιβλιοθήκη MULAN), που οφείλουν την επιτυχία τους στο ότι ανιχνεύουν και εκμεταλλεύονται τυχόν συσχετίσεις μεταξύ των μεταβλητών εξόδου.
  - ✓ Έτσι καταφέρνουν να πετύχουν μεγαλύτερη ακρίβεια από την απλή προσέγγιση η οποία αντιμετωπίζει την πρόβλεψη της κάθε μεταβλητής εξόδου ως ένα ανεξάρτητο πρόβλημα μάθησης και το επιλύει χρησιμοποιώντας κάποιον από τους συνήθεις αλγορίθμους μάθησης

Ημι-επιβλεπόμενη Μάθηση (Semi-supervised learning)

- Η κύρια διαφορά μεταξύ της επιβλεπόμενης μάθησης και της μη επιβλεπόμενης είναι ότι στην πρώτη, τα δεδομένα έχουν μια ετικέτα (*label*) και συνεπώς ανήκουν σε μια κλάση, ενώ στη δεύτερη δεν έχουν και επαφίεται στον αλγόριθμο να ανακαλύψει υπάρχουσες συσχετίσεις ή ομάδες (*πρότυπα*).
- Στην ημι-επιβλεπόμενη μάθηση (*semi-supervised learning*) οι αλγόριθμοι εκπαιδεύονται σε έναν συνδυασμό δεδομένων με και χωρίς ετικέτα.
- Αυτό είναι απαραίτητο σε περιπτώσεις όπου έχουμε πρόσβαση σε πολλά δεδομένα αλλά το κόστος (χρονικό, οικονομικό) για προσθήκη ετικέτας σε όλα είναι απαγορευτικό
  - ✓ για παράδειγμα σε ένα σύστημα ταξινόμησης εικόνων, εγγράφων ή σε ένα σύστημα αναγνώρισης ομιλίας.
  - ✓ Επιπλέον η προσθήκη ετικέτας από κάποιον ειδικό εκτός από το κόστος προσθέτει και κάποιο βαθμό υποκειμενικότητας.
- Μια απλή μέθοδος ημι-επιβλεπόμενης μάθησης είναι η αυτο-εκπαίδευση (self-training) κατά την οποία ένας ταξινομητής εκπαιδεύεται από τα υπάρχοντα χαρακτηρισμένα (δηλαδή με ετικέτα) δεδομένα και στη συνέχεια χρησιμοποιείται για να ταξινομήσει τα υπόλοιπα (τα μη χαρακτηρισμένα).
- Οι ταξινομητές συνήθως προβλέπουν την κλάση που ανήκει ένα άγνωστο δεδομένο με κάποια πιθανότητα.
- Οπότε από τα ταξινομημένα δεδομένα επιλέγουμε αυτά για τα οποία ο ταξινομητής δίνει τη μεγαλύτερη πιθανότητα (βεβαιότητα) και τα προσθέτουμε στο αρχικό σύνολο χαρακτηρισμένων δεδομένων.
- Στη συνέχεια ο ταξινομητής εκπαιδεύεται στο καινούριο σύνολο δεδομένων και η διαδικασία συνεχίζεται επαναληπτικά.
- Python (sklearn) module: sklearn.semi_supervised
  - ✓ https://scikit-learn.org/1.5/modules/semi_supervised.html

Ενεργή μάθηση (Active learning)

- Είναι μια περίπτωση ημι-επιβλεπόμενης μάθησης αλλά εδώ ο αλγόριθμος μάθησης μπορεί να επιλέξει τα δεδομένα για τα οποία θέλει να μάθει την ετικέτα τους και στη συνέχεια να τα χρησιμοποιήσει για την επανεκπαίδευση του.
- Η ενεργή μάθηση είναι μια επαναληπτική διαδικασία όπου ένας κλασσικός αλγόριθμος ταξινόμησης εκπαιδεύεται από τα δεδομένα με ετικέτα και στη συνέχεια το μοντέλο ταξινομεί τα δεδομένα χωρίς ετικέτα, προφανώς με μεγάλη πιθανότητα λάθους σε αρκετά από αυτά.
- Στη συνέχεια ένας μηχανισμός επιλογής, επιλέγει από τα ταξινομημένα δεδομένα ποια από αυτά θεωρεί σωστά ή αν θα ρωτήσει τον ειδικό για επιβεβαίωση.
- Η εκπαίδευση επαναλαμβάνεται λαμβάνοντας υπόψη και τα νέα δεδομένα και ελέγχεται η ακρίβεια του μοντέλου σε ένα σύνολο δοκιμής.
- Γενικά υπάρχουν πολλές πολιτικές επιλογής δεδομένων.
- Επίσης αντί για ένα μοντέλο μπορεί να χρησιμοποιηθεί συνδυασμός μοντέλων.
- scikit-activeml: A Library and Toolbox for Active Learning Algorithms
  - ✓ scikit-activeml has been developed as a Python library for active learning on top of scikit-learn

Εξ'αποστάσεως επιβλεπόμενη μάθηση (Distant supervised learning)

- Είναι επίσης περίπτωση ημι-επιβλεπόμενης μάθησης, όπου χρησιμοποιείται μια ευρετική συνάρτηση που συνήθως προκύπτει από ένα προκαθορισμένο σύνολο κανόνων, για να προστεθεί ετικέτα και στα δεδομένα που δεν έχουν, προφανώς με κάποιο ποσοστό ακρίβειας (αποδοχή ύπαρξης θορύβου).
  - ✓ Unlabeled data is heuristically/automatically labeled.
- Χρησιμοποιώντας τα (λίγα) αρχικά δεδομένα εκπαίδευσης και τα (θορυβώδη) δεδομένα που προέκυψαν, εκπαιδεύεται στη συνέχεια ο αλγόριθμος ταξινόμησης για την παραγωγή του τελικού μοντέλου πρόβλεψης.
- https://notesonai.com/distant+supervision

Learning-to-Rank

- Learning to rank (LtR) is the application of machine learning, typically supervised, in the construction of ranking models.
  - Training data may consist of lists of items with some partial order specified between items in each list.
  - This order is typically induced by giving a numerical or ordinal score or a binary judgment (e.g. "relevant" or "not relevant") for each item.
  - The goal is to rank new, unseen lists in a similar way to rankings in the training data.
- Applications
  - Information Retrieval: sorting documents by relevance with respect to a query
  - Search Engines: Given a user profile and a textual query, sort web pages results by relevance
  - Recommender Systems: Given a user profile and purchase history, sort the other items potentially interesting for the user
- Approaches
  - Direct LtR: Train a regression model to predict the relevance scores
  - Pairwise LtR: Train a binary model that receives two examples and predicts which is more relevant
  - Listwise LtR: Rank all data/documents simultaneously using loss functions that depend only on the order of data/documents
- The most common Performance metric is Normalized Discounted Cumulative Gain (NDCG).

$$\mathrm{CG_p} = \sum_{i=1}^{p} rel_i$$

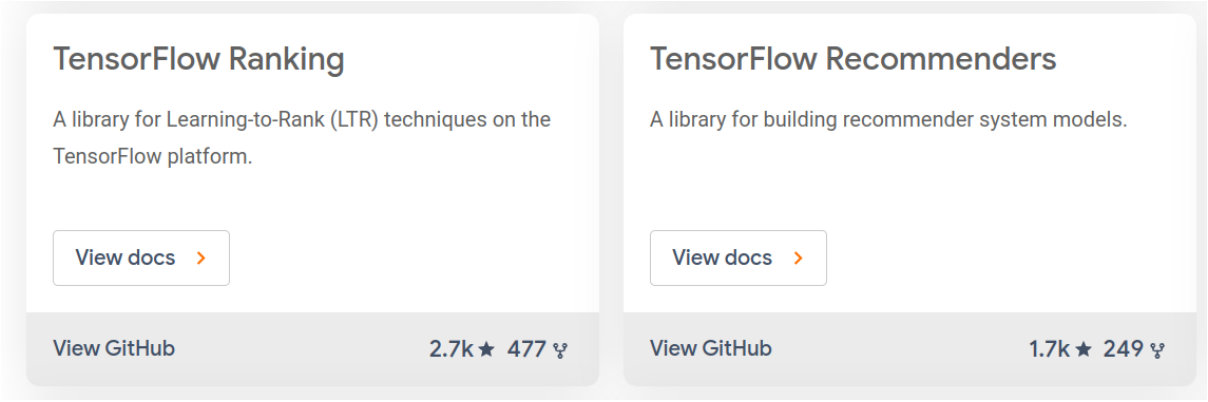
Where rel_i is the graded relevance of the result at position i.

- ✓ https://en.wikipedia.org/wiki/Learning_to_rank,
- ✓ Learning to Rank: A Complete Guide to Ranking using Machine Learning

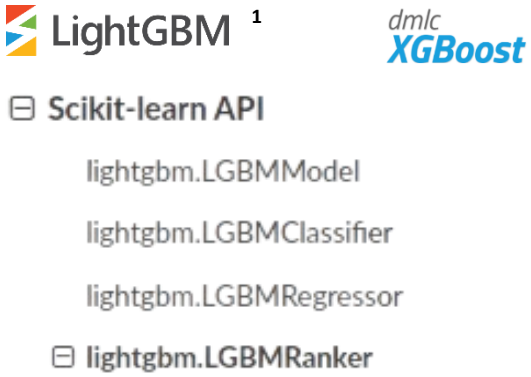
Frameworks for Learning-to-Rank

Tensorflow, LightGBM and XGBoost

Deep Learning



Gradient Boosting



- **Algorithms**: RankNet, LambdaRank, LambdaMART, ListNet, ListMLE
  - ✓ LambdaMart algorithm, is a Gradient boosting and can be implemented with XGBoost library or as Neural LTR using Tensor Flow
  - ✓ From RankNet to LambdaRank to LambdaMART: An Overview

[1] Εχει αναφερθεί σε προηγούμενο slide, στις XGBoost Libraries

…the end

Questions?