ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

2η ΕΡΓΑΣΙΑ ΜΑΘΗΜΑΤΟΣ «**ΤΕΧΝΙΚΕΣ ΕΞΟΡΥΞΗΣ ΔΕΔΟΜΕΝΩΝ**»

ΣΚΟΥΡΑΣ ΘΕΟΔΩΡΟΣ Α.Μ.: 1115 2013 00279

ΑΝΑΓΝΩΣΤΟΠΟΥΛΟΣ ΘΕΟΔΩΡΟΣ Α.Μ.: 1115 2014 00009

ΗΜΕΡΟΜΗΝΙΑ ΠΑΡΑΔΟΣΗΣ: 7 / 6 / 2017

Εισαγωγή

Η εργασία εκπονήθηκε σε Python 3 και με τη χρήση των εφαρμογών Pycharm και Jupyter Notebook. Ως εκ τούτου, τα αρχεία κώδικα που παράχθηκαν για τις ανάγκες της εργασίας και παραδίδονται είναι .py, αλλά το main αρχείο παραδίδεται και σε notebook file (.ipynb) για ευκολία. Για την καλύτερη κατανόηση του κώδικα, έχει προβλεφθεί ώστε να περιέχονται επεξηγηματικά σχόλια ανα κομμάτι που εκτελεί κάποια συσκεκριμένη εργασία.

Η εργασία χωρίζεται σε τρία μέρη:

- Οπτικοποίηση των δεδομένων
- Classification
- Evaluation of Classifiers and Processing of testSet

Όλες οι λειτουργικότητες υλοποιήθηκαν σε ένα αρχείο, το “askisi2.py” (askisi2.ipynb σε Jupyter Notebook). Τα αρχεία αυτά χρησιμοποιούν σαν βοηθητικό αρχείο το “func.py”, το οποίο περιέχει κάποιες συναρτήσεις που υλοποιήθηκαν από εμάς αλλά επιλέχθηκαν να μην εμφανίζονται στο main αρχείο για πρακτικούς/αισθητικούς λόγους. Για την σωστή λειτουργία των προγραμμάτων, έχει υποτεθεί ότι όλα τα αρχεία θα βρίσκονται στο ίδιο folder μαζί με τα αρχεία εισόδου δεδομένων (train.tsv και test.tsv).
Επίσης, στο ίδιο folder θα παραχθούν και τα αρχεία εξόδου που ζητούνται από την άσκηση.

Προεπεξεργασία των δεδομένων

Λόγω της ανομοιομορφίας των δεδομένων μας υπήρχε η ανάγκη κάποιας προεπεξεργασίας τους. Τα περισσότερα από τα attributes ήταν categorical, αλλά κάποια από αυτά ήταν numerical. **Αποφασίστηκε να τα μετετρέψουμε όλα σε categorical**, ελπίζοντας ότι μια μεγαλύτερη ομοιομορφία θα φέρει καλύτερα αποτελέσματα (παρόλα αυτά δοκιμάστηκε και να αφεθούν ανεπεξέργαστα τα numerical, με ελάχιστα χειρότερα accuracies σαν αποτέλεσμα).

Κατά την δημιουργία κατηγοριών των numerical, κρατήθηκε η τυποποίηση των υπόλοιπων categorical του dataset. Για παράδειγμα, το **A**ttribute**2** χωρίστηκε σε 5 κατηγορίες οι οποίες ονομάστηκαν **A2**1, **A2**2 κοκ. Τα 4 από τα 7 numerical ήταν ηδη "καβαντισμένα" σε 2 έως 4 κατηγορίες οπότε ήταν προφανής η κατηγοριοποίησή τους. Για τα υπόλοιπα 3, τα οποία είχαν μεγάλος πλήθος πιθανών τιμών (ηλικία, ποσό δανείου και διάρκεια δανείου) δημιουργήθηκαν 5 κατηγορίες "από-έως". Οι κατηγορίες δημιουργήθηκαν έτσι ώστε να είναι σχετικά ομοιόμορφα μοιρασμένα τα δεδομένα, αλλά και με κριτήριο το να υπάρχει όσο το δυνατό μεγαλύτερη διαφορά μεταξύ goods και bads ανά κατηγορία. Σε αυτό βοήθησαν τα barcharts που παρουσιάζονται στη συνέχεια αλλά και τα boxplots που μας έδειχναν σε ποιες κατηγορίες είναι συγκεντρωμένα τα περισσότερα goods και bads. Επίσης έγινε και μια τελική αξιολόγηση των κατηγοριών που δημιουργήθηκαν με τη βοήθεια του information gain που παρουσιάζεται στη συνέχεια (προσπαθήσαμε να πετύχουμε οσο μεγαλύτερο gain γινόταν ανα feature).

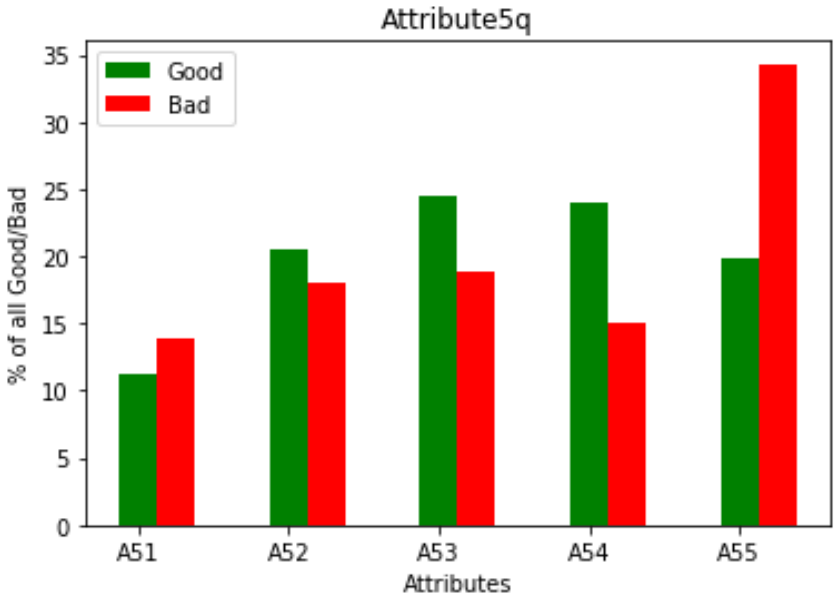
Για τη αποθήκευση των δεδομένων σε categorical δημιουργούσαμε για κάθε feature μια νέα στήλη η οποία είχε το ίδιο όνομα με το attribute με την κατάληξη "**q**". Η διαδικασία της μετατροπής μεταφέρθηκε στο αρχείο "func.py" οπότε στο main αρχείο φαίνεται μόνο η κλήση της συνάρτησης "convert_to_categorical " (η οποία μετατρέπει και τα 7 numerical σε categorical). Για παράδειγμα, το Attribute5 που αντιστοιχεί στο ποσό δανείου, αντιστοιχήθηκε στις κατηγορίες [<1000], [1001-1500], [1501-2500], [2501-4000], [>4000] με ονόματα ανα κατηγορία A51, A52, A53, A54, A55 και αποθηκεύθηκε στη στήλη Attribute5**q**. Ο κώδικας για το συγκεκριμένο είναι:

```
for i in range(0, len(dataset.Id)):
    if dataset.Attribute5[i] <= 1000:
        my_Attribute[i] = "A51"
    elif dataset.Attribute5[i] <= 1500:
        my_Attribute[i] = "A52"
    elif dataset.Attribute5[i] <= 2500:
        my_Attribute[i] = "A53"
    elif dataset.Attribute5[i] <= 4000:
        my_Attribute[i] = "A54"
    else:
        my_Attribute[i] = "A55"
dataset["Attribute5q"] = my_Attribute
```

Οπτικοποίηση των δεδομένων (Visualization)

Καθώς όλα μας τα δεδομένα υπήρχαν πια (και) σε categorical μορφή, επιλέχθηκε να οπτικοποιηθούν όλα σε bar charts. Συγκεκριμένα δημιουργήθηκε ένα bar chart ανα attribute, στο οποίο φαίνονται μια πράσινη μπάρα για τα good και μια κόκκινη για τα bad για κάθε κατηγορία του attribute. Επειδή η αναλογία σε good-bad ήταν 70%-30%, παρατηρούσαμε τα good εμφανώς περισσότερα για κάθε κατηγορία, κάτι το οποίο μας δυσκόλευε στο να βγάλουμε κάποιο συμπέρασμα για την κατανομή τους. Έτσι επιλέξαμε να κάνουμε normalization των δεδομένων μας, έτσι ώστε το αθροισμα των good σε όλες τις κατηγορίες του κάθε attribute, να είναι ίδιο με αυτό των bad (ίσο με 100). Έτσι μπορούσαμε να δούμε πλέον σε ποια κατηγορία του attribute είναι πιο πιθανό να δούμε έναν good ή έναν bad πελάτη.

Για παράδειγμα στο bar chart του attribute5q (το οποίο έχει μετατραπεί από εμάς σε categorical):

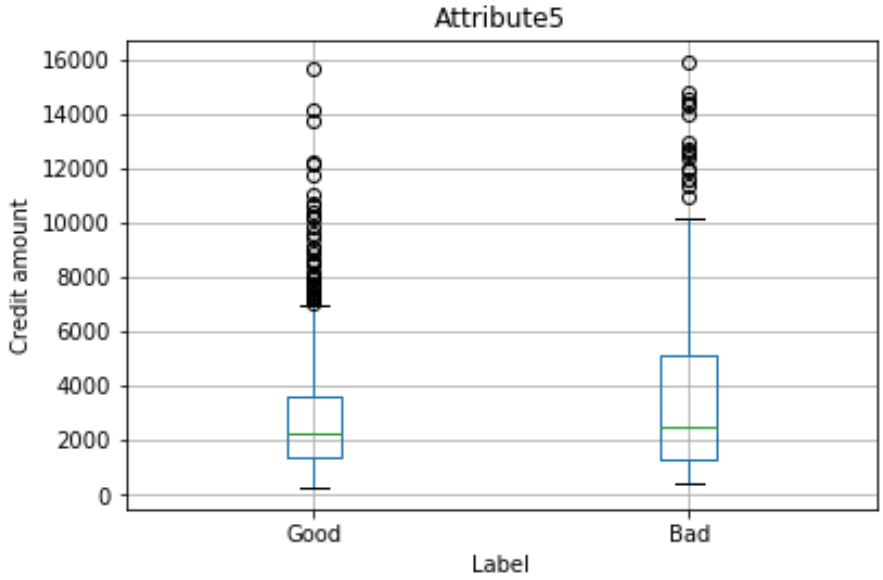


παρατηρούμε εύκολα ότι στην κατηγορία A55 (ποσό δανείου > 4000) είναι πιο πιθανό να βρούμε bad πελάτη, αφού σε αυτή την κατηγορία συγκεντρώνεται το 35% των bad αλλά μόνο το 20% των good.

*Σημειώνεται ότι η δυνατότητα του **normalization είναι optional** και μπορεί να **απενεργοποιηθεί** θέτοντας το flag "normalize" ίσο με 0.*

Από τα numerical attributes, επιλέξαμε για οπτικοποίηση σε box plots μόνο τα 3 που είχαν πληθώρα πιθανών τιμών, καθώς για τα υπόλοιπα δεν έβγαινε κάποιο ουσιαστικό συμπέρασμα λόγω του περιορισμένου πεδίου των τιμών τους (μόνο 2 - 4 πιθανές τιμές).

Για παράδειγμα στο Box plot του attribute5:



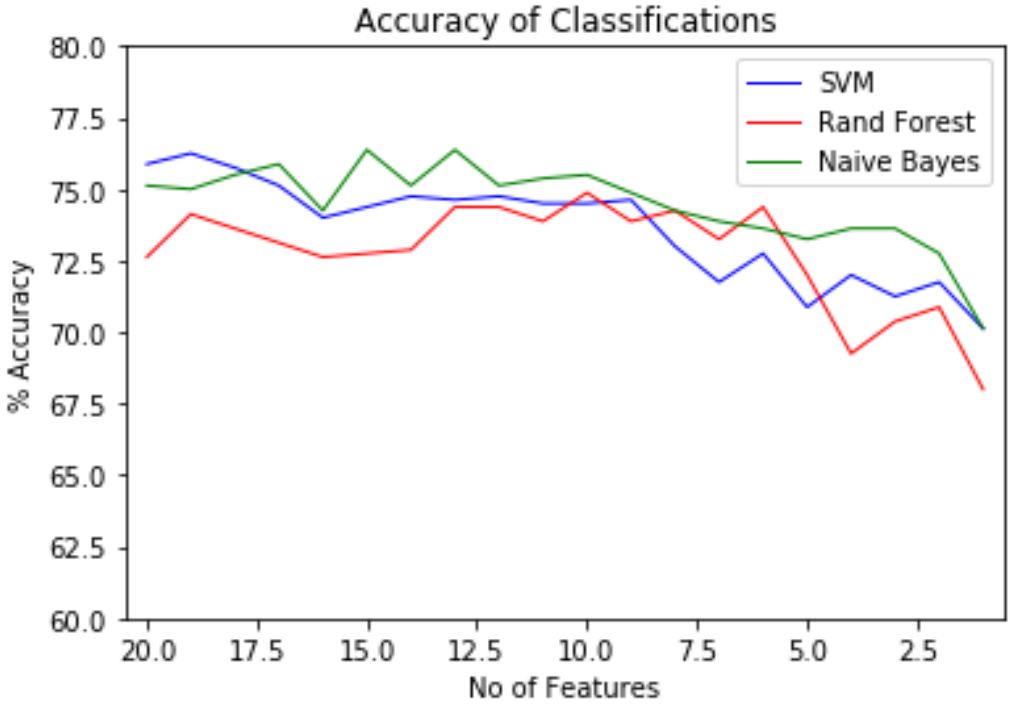
παρατηρούμε ότι οι good πελάτες συγκεντρώνονται κυρίως στα ποσά δανείων μεταξύ 1500 και λίγο λιγότερο από 4000, ενώ οι bad μεταξύ 1500 και περίπου 5000. Είναι λογικό λοιπόν να βγάλουμε το συμπέρασμα ότι η δημιουργία μιας κατηγορίας [>4000] θα είχε μεγαλύτερο ποσοστό σε bad πελάτες όπως είδαμε και στο προηγούμενο bar chart.

....

Υλοποίηση Κατηγοριοποίησης (Classification)

Η κατηγοριοποίηση έγινε με τους 3 τρόπους που ζητήθηκε και χρησιμοποιώντας την τεχνική (10fold) cross validation. Για την επιλογή των βέλτιστων χαρακτηριστικών του SVC χρησιμοποιήθηκε το "GridSearchCV" όπως και στην προηγούμενη άσκηση, το οποίο μετά μας έδωσε C=5, gamma=0.2, kernel='rbf'. Στον RandomForestClassifier επιλέξαμε να δημιουργούνται τόσα δένδρα απόφασης όσα και ο αριθμός των attributes που επεξεργάζεται κάθε φορά. Ο MultiNomialNB αφέθηκε να λειτουργήσει με την default συμπεριφορά του.

Η διαδικασία έγινε 20 φορές (για κάθε classifier), ξεκινώντας με όλα τα attributes και **αφαιρώντας κάθε φορά αυτό με το μικρότερο information gain**. Το mean accuracy του κάθε classifier για κάθε αριθμό attributes που επεξεργάστηκε φαίνεται παρακάτω:



Σημειώνεται ότι τα δεδομένα που επεξεργάστηκαν οι classifiers, προέκυψαν από την get_dummies του pandas. Συγκεκριμένα, η διαδικασία που ακολουθήσαμε ήταν η εξής:

- Υπολογισμός information gain για κάθε attribute (έγινε στο βήμα κατασκευής των bar charts με την συνάρτηση "info_gain" η υλοποίησητης οποίας υπάρχει στο "func.py")
- Sort των 20 (categorical) attributes κατά αύξουσα σειρά information gain
- Επιλογή των attributes με το μεγαλύτερο information gain κάθε φορά και τροφοδότηση όλων στην get_dummies
- Μετατροπή των categorical από την get_dummies, σε έναν ενιαίο πίνακα ανα πελάτη ο οποίος περιείχε τόσα στοιχεία, όσα όλες οι διαφορετικές κατηγορίες όλων των attributes, με "1" και "0" σαν πιθανές τιμές (τόσα "1" όσα ο αριθμός των attributes κάθε φορά).
- Τροφοδότηση των classifiers με τα output από την get_dummies
- Υπολογισμός μέσου accuracy (των 10 folds) για κάθε classifier για κάθε αριθμό επεξεργασθέντων attributes (20 έως 1).

Επιλογή Βέλτιστου Classifier/αριθμού Attributes και Process του testSet

Για την επεξεργασία του testSet, επιλέγεται **αυτόματα** από το πρόγραμμα:

- **ο classifier** με το καλύτερο μέσο (για τις 20 εκτελέσεις από 20 έως 1 attributes) mean accuracy**
- **ο αριθμός των attributes** που παράγει καλύτερο accuracy** κατά μέσο όρο για όλους τους classifiers

((Θεωρούσαμε πιο σωστό να υλοποιήσουμε ενα cost_matrix ωστε να βγει ενα weighted score , επειδή με το απλό mean accuracy, τα False Positives εχουν το ιδιο impact/βαρος με τα False Negatives.
Αλλα δεν ζητήθηκε αυτο, από πλευρά εκφώνησης, παρόλο που η προβλεψη ενός BAD ως Good ειναι σοβαρό λαθος.))

Για τον βέλτιστο αριθμό attributes επαναλαμβάνεται η διαδικασία του classification και παράγεται και εκτυπώνεται το “EvaluationMetric_10fold.csv”.

Stat/Meas	Naive Bayes	Random Forest	SVM
0 Accuracy	0.75375	0.7575	0.7475

Για τον βέλτιστο αριθμό attributes και χρησιμοποιώντας τον βέλτιστο classifier, επεξεργαζόμαστε το αρχείο “testset.tsv” παράγοντας την πρόβλεψή μας για good ή bad ανα πελάτη στο αρχείο “testSet_Predictions.csv” (επίσης εκτυπώνεται). Η διαδικασία, βρίσκεται στο “func.py”, με την main να καλεί την συνάρτηση “classify_test()”.

```
classifying 15 feat. with MultinomialNB(alpha=1.0,class_prior=None,fit_prior=True)
     Client_ID Predicted_Category
0        10902               Good
1        10903                Bad
2        10904               Good
3        10905                Bad
4        10906                Bad
5        10907               Good
6        10908               Good
7        10909                Bad
     ...
```

Σημειώνεται, ότι όπως φαίνεται και παραπάνω, καθώς επιλέγεται ο classifier που έχει κατά μέσο όρο τη βέλτιστη συμπεριφορά και ο αριθμός attributes επίσης, μπορεί να παρατηρηθεί το φαινόμενο ο classifier που επιλέχθηκε να έχει ελαφρώς μικρότερο accuracy για το συγκεκριμένο instance αριθμού attributes από κάποιον άλλο. Για παράδειγμα, ο NaiveBayes που επιλέχθηκε στην περίπτωσή μας, έχει accuracy = 0,75375 ενώ ο Random Forest 0.7575.

==========