Μπαζάκας Τηλέμαχος Μάρκος 3281

Ντόντης Βασίλειος 3300

Μηχανική Μάθηση

1η Σειρά Ασκήσεων

Θέμα: Μέθοδοι ταξινόμησης ή κατηγοριοποίησης δεδομένων (Machine Learning classification methods)

Πανεπιστήμιο Ιωαννίνων

Στόχος της εργασίας είναι να μελετήσουμε πειραματικά την επίδοση γνωστών αλγορίθμων Μηχανικής Μάθησης στο πρόβλημα της ταξινόμησης. Χρησιμοποιήσαμε τις έτοιμες μεθόδους ταξινόμησης της sk-learn βιβλιοθήκης. Για κάθε μεθοδολογία υϊοθετούμε την γνωστή τεχνική του 10-fold cross validation για την αντιμετώπιση του overfitting και της αύξησης της γενίκευσης του εκάστοτε ταξινομητή. Οι μέθοδοι ταξινόμησης που μελετήσαμε είναι οι:

- ➔ k-NN Nearest Neighbors (για k=1, 3, 5 και 10 γείτονες)
- ➔ Naïve Bayes classifier
- ➔ Neural Networks
- ➔ Support Vector Machines (SVM)

Στη διάθεση μας έχουμε δύο σύνολα δεδομένων στα οποία μελετήσαμε τις παραπάνω μεθόδους ταξινόμησης. Κάθε σύνολο δεδομένων, το χωρίζουμε (τυχαία) σε δύο υποσύνολα για μάθηση και έλεγχο σε αναλογία 70-30. Εν τέλει, εφαρμόζουμε την κάθε μέθοδο ταξινόμησης ξεχωριστά.

Διαδικασία κατασκευής μεθόδων

K Nearest Neighbors - (kNN) - Mobile Price train set

Αρχικά, διαβάζουμε το csv αρχείο, κρατάμε τα δεδομένα στο Dataframe ‘data’, και χωρίζουμε τις στήλες με διακριτές τιμές και συνεχές στους πίνακες features_to_move και features to stay. Αλλάζουμε το data, έτσι ώστε να έχει πρώτα τις στήλες με διακριτές τιμές και μετά αυτές με τις συνεχείς. Και γεμίζουμε τους πίνακες discrete_features και continuous features με τις θέσεις των αντίστοιχων τιμών στο Dataframe ‘data’. Στην συνέχεια αποθηκεύουμε στην μεταβλητή y το Dataframe με την στήλη για τις κλάσεις και την αφαιρούμε από το x.

```
data = pd.read_csv('train.csv')
features_to_move = ['blue', 'clock_speed', 'dual_sim', 'fc', 'four_g', 'int_memory', 'm_dep',
                    'n_cores', 'pc', 'sc_h', 'sc_w', 'talk_time', 'three_g', 'touch_screen', 'wifi']
features_to_stay = [col for col in data.columns if col not in features_to_move]
data = data[features_to_move + features_to_stay]

# make indexes for each feature
discrete_features = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
continuous_features = [15, 16, 17, 18, 19]

target_col = 'price_range'
y = data[target_col]

x = data.drop(target_col, axis=1)
```

Στην συνέχεια χωρίζουμε το dataset τυχαία σε δύο υποσύνολα για μάθηση και αξιολόγηση, 70% για μάθηση και 30% για αξιολόγηση. Εκτελούμε την εντολή to_numpy() για να μπορούμε να διαχειριστούμε στην συνέχεια τα X και Y. Για να πραγματοποιήσουμε την διασταύρωση 10-Fold με 10 διασταυρώσεις, εκτελούμε την εντολή ‘kf = KFold(n_splits=10)’, έτσι μπορούμε να χωρίσουμε τα δεδομένα σε διασταυρώσεις και να εκπαιδεύσουμε και να αξιολογήσουμε το μοντέλο μηχανικής μάθησης σε κάθε διασταύρωση. Τέλος αρχικοποιούμε τους πίνακες που θα αποθηκευτούν εντός του τα F1-scores και τα accuracies.

```
X, X_, Y, Y_ = train_test_split(x, y, test_size=0.3, random_state=42)
X = X.to_numpy()
Y = Y.to_numpy()

kf = KFold(n_splits=10)

# Initialize empty lists to store the F1-score and accuracy for each fold
f1_scores = []
accuracies = []

f1_scores2 = []
accuracies2 = []
```

Έπειτα δημιουργούμε την συνάρτηση custom_distance, καθώς πρέπει να υποθέσουμε Ευκλείδια απόσταση για τις συνεχείς μεταβλητές και απόσταση Hamming για τις διακριτές μεταβλητές και η απόσταση προκύπτει από το άθροισμα των δύο παραπάνω αποστάσεων. Η απόσταση Hamming είναι μια μετρική που χρησιμοποιείται για να μετρήσει την απόσταση μεταξύ δύο αλφαριθμητικών συμβολοσειρών ίδιου μήκους. Στη δική μας συνάρτηση, μέσω της εντολής 'np.sum(x[:len(features_to_move)-1] != y[:len(features_to_move)-1])' υπολογίζουμε τον αριθμό των χαρακτηριστικών που δεν είναι ίδια μεταξύ του 'x' και του 'y'. Εφόσον τα διακριτά χαρακτηριστικά έχουν δυαδικές τιμές (δηλαδή 0 ή 1), αυτός ο υπολογισμός μετρά τον αριθμό των χαρακτηριστικών όπου τα x και y έχουν διαφορετικές δυαδικές τιμές. Έτσι, η απόσταση Hamming μετράει πόσα δυαδικά χαρακτηριστικά διαφέρουν μεταξύ του 'x' και του 'y' δείγματος. Η Ευκλείδεια απόσταση είναι μια μετρική που χρησιμοποιείται για να μετρήσει την απόσταση μεταξύ δύο σημείων σε ένα πολυδιάστατο χώρο. Στη συνάρτησή μας, με την χρησιμοποίηση της εντολής 'x[len(features_to_move)-1:] - y[len(features_to_move)-1:]' αφαιρούμε τις συνεχείς τιμές χαρακτηριστικών του 'y' από το 'x', και με την χρήση του 'np.linalg.norm()' υπολογίζουμε το μήκος του αποτελεσματικού διανύσματος. Έτσι, η ευκλείδεια απόσταση μετρά το μήκος του διανύσματος που συνδέει τα δύο σημεία στον χώρο των συνεχών χαρακτηριστικών. Τελικά, η συνάρτηση επιστρέφει το άθροισμα της απόστασης Hamming και της ευκλείδειας απόστασης ως συνολική απόσταση μεταξύ των δύο δειγμάτων.

```
def custom_distance(x, y):
    # Compute Hamming distance for discrete features
    hamming_dist = np.sum(x[:len(features_to_move)-1] != y[:len(features_to_move)-1])
    # Compute Euclidean distance for continuous features
    euclidean_dist = np.linalg.norm(x[len(features_to_move)-1:] - y[len(features_to_move)-1:])
    # Return the combined distance
    return hamming_dist + euclidean_dist
```

Στην συνέχεια τρέχουμε την μέθοδο για τιμές του K (γείτονες) 1,3,5 και 10. Αρχικά δημιουργούμε τον classifier με k γείτονες και ως παράμετρο την συνάρτηση που δημιουργήσαμε για τον υπολογισμό των αποστάσεων. Δημιουργούμε το αντικείμενο kf για την πραγματοποίηση της διασταύρωσης και αρχικοποιούμε τους πίνακες που θα κρατήσουν τα αποτελέσματα. Για κάθε διασταύρωση (fold), χωρίζουμε τα δεδομένα σε training και validation sets και για τα X και για τα Y. Κάνουμε fit τα δεδομένα που είχαμε ορίσει για train, δηλαδή το 70% των δεδομένων, και κάνουμε predict για το υπόλοιπο 30% των δεδομένων που είναι για validation. Τοποθετούμε τα αποτελέσματα F1-score και accuracy στους πίνακες f1_scores και accuracies, συγκρίνοντας τα predicted με τα validation δεδομένα. Τέλος, για κάθε τιμή που αναθέτουμε στο K, δηλαδή τον αριθμό γειτόνων, γίνεται και το αντίστοιχο τύπωμα στον χρήστη, που περιέχει τον αριθμό των γειτόνων και τα αποτελέσματα αυτής της μεθόδου.

```python
for k in [1, 3, 5, 10]:
    # Create a KNeighborsClassifier object with k=3 and the custom metric
    knn = KNeighborsClassifier(n_neighbors=k, metric=custom_distance)

    kf = KFold(n_splits=10)

    # Initialize empty lists to store the F1-score and accuracy for each fold
    f1_scores = []
    accuracies = []
    # Loop over each fold
    for train_index, test_index in kf.split(X):
        # Split the data into training and test sets for this fold
        X_train, X_val = X[train_index], X[test_index]
        y_train, y_val = Y[train_index], Y[test_index]

        # Fit the KNN classifier on the training data
        knn.fit(X_train, y_train)

        # Predict the test labels using the trained KNN classifier
        y_pred = knn.predict(X_val)

        # Calculate the F1-score and accuracy for this fold
        f1_scores.append(f1_score(y_val, y_pred, average='macro'))
        accuracies.append(accuracy_score(y_val, y_pred))

    # Print the F1-score and accuracy for each fold
    for i in range(len(f1_scores)):
        print(f"Fold {i+1}: F1-score = {f1_scores[i]:.4f}, Accuracy = {accuracies[i]:.4f}")
    print("For", k, "neighbors the average F1-score is:", sum(f1_scores)/len(f1_scores),
          "and the average Accuracy is:", sum(accuracies)/len(accuracies))
```

Naïve Bayes classifier - Mobile Price train set

Εκτελούμε τον ίδιο κώδικα για την διαχείριση των δεδομένων αφού διαβάσουμε τα δεδομένα από το csv. Επίσης δημιουργούμε δύο πίνακες με δείκτες στα features που έχουν διακριτά και συνεχή δεδομένα.

Στο κομμάτι της μάθησης, υποθέτουμε (ανεξάρτητη) κανονική κατανομή (normal distribution) για κάθε ένα από τα χαρακτηριστικά συνεχούς τιμής και πολυωνυμική (multinomial distribution) κατανομή για τα διακριτά χαρακτηριστικά και αντίστοιχα κάνουμε fit για το 70% των δεδομένων που χωρίσαμε για train, για τις διακριτές και τις συνεχείς τιμές ξεχωριστά. Στη συνέχεια, δημιουργούμε μεταβλητή για κάθε fit που κάναμε όπου κρατάμε τις predicted πιθανότητες για τις διακριτές και τις συνεχείς τιμές ξεχωριστά. Τελικά πολλαπλασιάζουμε τις δύο αυτές μεταβλητές για να συνενώσουμε τις πιθανότητες και τέλος, το y_pred κρατάει τις τη μεγαλύτερη πιθανότητα για κάθε class.

```
# Loop over each fold
for train_index, test_index in kf.split(X):
    # Split the data into training and test sets for this fold
    X_train, X_val = X[train_index], X[test_index]
    y_train, y_val = Y[train_index], Y[test_index]

    # Fit the KNN classifier on the training data
    multi_clf = MultinomialNB()
    multi_clf.fit(X_train[:, discrete_features], y_train)

    gauss_clf = GaussianNB()
    gauss_clf.fit(X_train[:, continuous_features], y_train)

    multi_probs = multi_clf.predict_proba(X_val[:, discrete_features])
    gauss_probs = gauss_clf.predict_proba(X_val[:, continuous_features])

    # Combine the probabilities using the product rule
    probs = np.multiply(multi_probs, gauss_probs)

    # Predict the class with the highest probability
    y_pred = np.argmax(probs, axis=1)
```

Στη συνέχεια ακολουθούμε την ίδια διαδικασία υπολογισμού του F1-score και του accuracies, με τις προηγούμενες μεθόδους ταξινόμησης.

Neural Networks - Mobile Price train set

Εκτελούμε τον ίδιο κώδικα για την διαχείριση των δεδομένων αφού διαβάσουμε τα δεδομένα από το csv.

Στο κομμάτι της μάθησης, δημιουργούμε δύο ξεχωριστούς classifiers:

(α)έναν με 1 κρυμμένο επίπεδο και K κρυμμένους νευρώνες, και

(β)έναν με 2 κρυμμένα επίπεδα αποτελούμενο από K1 και K2 νευρώνες, αντίστοιχα.

Σε κάθε classifier, χρησιμοποιούμε σιγμοειδή συνάρτηση ενεργοποίησης στους κρυμμένους νευρώνες (sigmoid activation function) sigmoid or tanh και χρησιμοποιούμε τη συνάρτηση ενεργοποίησης softmax όπως μας ζητείται. Οι τιμές που χρησιμοποιήσαμε για τους classifiers είναι: (100,) και (100,50) για 1 κρυμμένο επίπεδο και K κρυμμένους νευρώνες και 2 κρυμμένα επίπεδα και K1 και K2 κρυμμένους νευρώνες αντίστοιχα. Τελικά, κάνουμε fit και predict όπως ακριβώς και στις προηγούμενες μεθόδους ταξινόμησης για τα 70-30 χωρισμένα δεδομένα.

```
# Loop over each fold
for train_index, test_index in kf.split(X):
    # Split the data into training and test sets for this fold
    X_train, X_val = X[train_index], X[test_index]
    y_train, y_val = Y[train_index], Y[test_index]

    clf = MLPClassifier(solver='sgd', hidden_layer_sizes=(100,), activation='tanh')
    # Set the output activation function to "softmax"
    clf.out_activation_ = 'softmax'
    # Train the classifier on the training set
    clf.fit(X_train, y_train)

    y_pred = clf.predict(X_val)

    clf2 = MLPClassifier(solver='sgd', hidden_layer_sizes=(100, 50), activation='tanh')
    # Set the output activation function to "softmax"
    clf2.out_activation_ = 'softmax'
    # Train the classifier on the training set
    clf2.fit(X_train, y_train)

    y_pred2 = clf2.predict(X_val)
```

Στη συνέχεια ακολουθούμε την ίδια διαδικασία υπολογισμού του F1-score και του accuracies, με τις προηγούμενες μεθόδους ταξινόμησης.

Support Vector Machines - (SVM) - Mobile Price train set

Εκτελούμε τον ίδιο κώδικα για την διαχείριση των δεδομένων αφού διαβάσουμε τα δεδομένα από το csv. Για κάθε διασταύρωση χωρίζουμε τα δεδομένα σε training και validation sets και για τα X και για τα Y και δημιουργούμε ξεχωριστό classifier για:

(α) γραμμική συνάρτηση πυρήνα (linear kernel) - clf, και
(β) gaussian συνάρτηση πυρήνα RBF (kernel) – clf2.

Και στις δύο περιπτώσεις χρησιμοποιούμε την στρατηγική one-versus-all μέσω της παραμέτρου ‘decision_function_shape’.

Στην συνέχεια κάνουμε fit τα δεδομένα που είχαμε ορίσει για train, δηλαδή το 70% των δεδομένων, και κάνουμε predict για το υπόλοιπο 30% των δεδομένων που είναι για validation. Εκτελούμε την ίδια διαδικασία και για τον clf2 που είναι για την gaussian συνάρτηση και τέλος τοποθετούμε τα αποτελέσματα F1-score και accuracy στους πίνακες f1_scores και accuracies, συγκρίνοντας τα predicted με τα validation δεδομένα.

```
# Loop over each fold
for train_index, test_index in kf.split(X):
    # Split the data into training and test sets for this fold
    X_train, X_val = X[train_index], X[test_index]
    y_train, y_val = Y[train_index], Y[test_index]

    clf = svm.SVC(kernel='linear', decision_function_shape='ovr')

    clf.fit(X_train, y_train)

    y_pred = clf.predict(X_val)

    clf2 = svm.SVC(kernel='rbf', C=1.0, gamma=0.0000001, decision_function_shape='ovr')

    clf2.fit(X_train, y_train)

    y_pred2 = clf2.predict(X_val)

    # Calculate the F1-score and accuracy for this fold
    f1_scores.append(f1_score(y_val, y_pred, average='macro'))
    accuracies.append(accuracy_score(y_val, y_pred))

    f1_scores2.append(f1_score(y_val, y_pred2, average='macro'))
    accuracies2.append(accuracy_score(y_val, y_pred2))
```

Τέλος, τυπώνουμε στον χρήστη τα αποτελέσματα για όλα τα Folds, αλλά και τον μέσο όρο τον αποτελεσμάτων.

```
# Print the F1-score and accuracy for each fold
for i in range(len(f1_scores)):
    print(f"Fold {i+1}: F1-score = {f1_scores[i]:.4f}, Accuracy = {accuracies[i]:.4f}")
print("The average F1-score is:", sum(f1_scores)/len(f1_scores), "and the average Accuracy is:",
      sum(accuracies)/len(accuracies))


for i in range(len(f1_scores2)):
    print(f"Fold {i+1}: F1-score = {f1_scores2[i]:.4f}, Accuracy = {accuracies2[i]:.4f}")
print("The average F1-score is:", sum(f1_scores2)/len(f1_scores2), "and the average Accuracy is:",
      sum(accuracies2)/len(accuracies2))
```

Airlines Delay Code

Ο κώδικας που χρησιμοποιήθηκε για τα αρχεία του Mobile Price train set, είναι παρόμοιος με τον κώδικα που χρησιμοποιήθηκε για τα αρχεία του Airlines Delays. Η μόνη διαφορά είναι η διαχείριση των δεδομένων. Παρακάτω δείχνουμε τις διαφορές σε κώδικα.

```
le = LabelEncoder()
data['Airline'] = le.fit_transform(data['Airline'])
data['AirportFrom'] = le.fit_transform(data['AirportFrom'])
data['AirportTo'] = le.fit_transform(data['AirportTo'])

delete_col = 'Flight'
target_col = 'Class'
y = data[target_col]
z = data[delete_col]

x = data.drop(target_col, axis=1)
x = x.drop(delete_col, axis=1)
```

Χρησιμοποιήσαμε έναν LabelEncoder, για να μετατρέψουμε τις αλφαριθμητικές μεταβλητές των στηλών (Airline, AirportFrom, AirportTo) σε διακριτές τιμές. Έπειτα αφαιρέσαμε την στήλη 'Flight', επειδή δεν μας ενδιαφέρουν τα δεδομένα της και όπως και στον κώδικα που χρησιμοποιήθηκε στο αρχείο Mobile Price, αφαιρούμε και την target στήλη από το 'x' και την τοποθετούμε στο 'y'.

Αποτελέσματα δοκιμών

Mobile Price - KNN - 1 neighbor

Fold	F1-score	Accuracy
1st	0.9365	0.9357
2nd	0.8881	0.8929
3rd	0.8580	0.8571
4th	0.8573	0.8571
5th	0.8968	0.9000
6th	0.9157	0.9143
7th	0.8783	0.8857
8th	0.8641	0.8643
9th	0.9021	0.9000
10th	0.8265	0.8286
Average	***0.8823346596124193***	***0.8835714285714287***

Mobile Price - KNN - 3 neighbors

Fold	F1-score	Accuracy
1st	0.9295	0.9286
2nd	0.9497	0.9500
3rd	0.8782	0.8786
4th	0.9225	0.9214
5th	0.9269	0.9286
6th	0.9574	0.9571
7th	0.9174	0.9214
8th	0.9065	0.9071
9th	0.9101	0.9071
10th	0.8851	0.8857
Average	***0.918318297315914***	***0.9185714285714287***

Mobile Price - KNN - 5 neighbors

Fold	F1-score	Accuracy
1st	0.9438	0.9429
2nd	0.9491	0.9500
3rd	0.9372	0.9357
4th	0.9294	0.9286
5th	0.9120	0.9143
6th	0.9435	0.9429
7th	0.9284	0.9286
8th	0.9150	0.9143
9th	0.9171	0.9143
10th	0.9073	0.9071
Average	***0.9282667013710013***	***0.9278571428571428***

Mobile Price - KNN - 10 neighbors

Fold	F1-score	Accuracy
1st	0.9508	0.9500
2nd	0.9493	0.9500
3rd	0.9464	0.9429
4th	0.9032	0.9000
5th	0.9272	0.9286
6th	0.9361	0.9357
7th	0.8965	0.9000
8th	0.9071	0.9071
9th	0.9247	0.9214
10th	0.9381	0.9357
Average	***0.9279386075829341***	***0.927142857142857***

Mobile Price - Bayes

Fold	F1-score	Accuracy
1st	0.7568	0.7500
2nd	0.7704	0.7786
3rd	0.7484	0.7429
4th	0.7658	0.7643
5th	0.8345	0.8357
6th	0.6777	0.6714
7th	0.6948	0.7071
8th	0.7964	0.7929
9th	0.8245	0.8214
10th	0.6649	0.6500
Average	***0.7534141203726241***	***0.7514285714285714***

Mobile Price - Neural - One hidden layer with 100 hidden neurons

Fold	F1-score	Accuracy
1st	0.5091	0.5214
2nd	0.5154	0.5429
3rd	0.4796	0.4929
4th	0.5220	0.5429
5th	0.5199	0.5286
6th	0.4970	0.5214
7th	0.5420	0.5786
8th	0.4755	0.4929
9th	0.5081	0.5286
10th	0.4530	0.4571
Average	***0.5021612348896596***	***0.5207142857142857***

Mobile Price - Neural - Two hidden layers with 100,50 hidden neurons

Fold	F1-score	Accuracy
1st	0.5353	0.5571
2nd	0.4672	0.4929
3rd	0.4277	0.4571
4th	0.4574	0.5286
5th	0.5487	0.5714
6th	0.5142	0.5357
7th	0.5130	0.5500
8th	0.5235	0.5429
9th	0.5578	0.5643
10th	0.4420	0.4500
Average	***0.4986770029965979***	***0.5249999999999999***

Mobile Price - SVM - Linear

Fold	F1-score	Accuracy
1st	0.9793	0.9786
2nd	0.9703	0.9714
3rd	0.9655	0.9643
4th	0.9526	0.9500
5th	0.9634	0.9643
6th	0.9567	0.9571
7th	0.9319	0.9357
8th	0.9559	0.9571
9th	0.9803	0.9786
10th	0.9777	0.9786
Average	***0.9633556002999942***	***0.9635714285714284***

Mobile Price - SVM - Gaussian RBF, C:1.0, gamma = near to 'scale'

Fold	F1-score	Accuracy
1st	0.9580	0.9571
2nd	0.9466	0.9500
3rd	0.9504	0.9500
4th	0.9355	0.9357
5th	0.9403	0.9429
6th	0.9640	0.9643
7th	0.9295	0.9357
8th	0.9114	0.9143
9th	0.9371	0.9357
10th	0.9650	0.9643
Average	***0.9437833458345228***	***0.945***

Airlines Delay – Bayes

Fold	F1-score	Accuracy
1st	0.5322	0.5342
2nd	0.5307	0.5327
3rd	0.5288	0.5307
4th	0.5298	0.5319
5th	0.5349	0.5368
6th	0.5316	0.5340
7th	0.5295	0.5312
8th	0.5343	0.5363
9th	0.5345	0.5361
10th	0.5280	0.5298
Average	***0.5314111825422018***	***0.5333702416927952***

Βέλτιστη Μέθοδος

Σύμφωνα με την σύγκριση των αποτελεσμάτων των μεθόδων, η μέθοδος με τα καλύτερα αποτελέσματα είναι η Support Vector Machines (SVM) χρησιμοποιώντας γραμμική συνάρτηση πυρήνα (linear kernel).

Παράρτημα κώδικα : https://github.com/MarkBaz/Machine-Learning-classification-methods