ΑΝΑΓΝΩΡΙΣΗ ΠΡΟΤΥΠΩΝ PROJECT 2018

ΣΤΑΜΑΤΟΠΟΥΛΟΣ ΒΑΣΙΛΕΙΟΣ

1115201400188

Στο κείμενο αυτό περιγράφεται η υλοποίηση των αλγορίθμων καθώς και οι μέθοδοι που χρησιμοποιήθηκαν με την εξής σειρά

1) Naïve Bayes
2) Euclidean Distance
3) K-nearest neighbors

Αφού γίνει προ-επεξεργασία των test και operation sets όπως και το train, ξεκινάει η υλοποίηση των αλγορίθμων που ζητούνται στην εργασία.

Naïve Bayes

Για το Naïve Bayes, αρχικά έτρεξα την εκδοχή της MATLAB για να συγκρίνω τα αποτελέσματα. Τα αποτελέσματα της ήταν τα παρακάτω

652	0	0	0	0
0	333	8	0	1
0	1	548	0	0
0	0	0	267	8
0	2	0	0	331

Ποσοστό επιτυχίας = 0.9843

Για την δικιά μου υλοποίηση, έδρασα με τον εξής τρόπο.

Βρήκα τις a-priori πιθανότητες για κάθε κλάση, δηλαδή το πόσο συχνά συναντάμε κάθε κλάση στα δεδομένα εκπαίδευσης. Έπειτα υπολόγισα το likelihood για κάθε κλάση με τη χρήση μιας normal cdf, και τέλος την a-posteriori πιθανότητα κάθε στοιχείο να ανήκει σε κάθε κλάση.

Μετά έγιναν predictions στο test-set και δημιουργήθηκε ο confusion matrix που είναι ο παρακάτω

652	0	0	0	0
1	0	341	0	0
0	0	547	2	0
0	0	0	263	12
0	0	0	12	335

Ποσοστό επιτυχίας = 0.8300

Όπως φαίνεται η υλοποίηση μου είχε προβλήματα στην κατηγοριοποίηση δεδομένων στην $2^{η}$ κλάση.

Euclidean Distance Classifier

Στη συνέχεια υλοποίησα τον ταξινομητή Ευκλείδειας Απόστασης.

Για κάθε στοιχείο στον πίνακα testing, ελέγχω την απόσταση του από κάθε στοιχείου του πίνακα εκπαίδευσης και βρίσκω την μικρότερη. Η πρόβλεψη είναι η κλάση στην οποία ανήκει αυτό το στοιχείο.

Για την υλοποίηση αυτή πήρα τον ακόλουθο πίνακα σύγχυσης.

652	0	0	0	0
0	337	5	0	0
0	0	549	0	0
0	0	0	271	4
0	0	0	9	338

Ποσοστό επιτυχίας = 0.9917

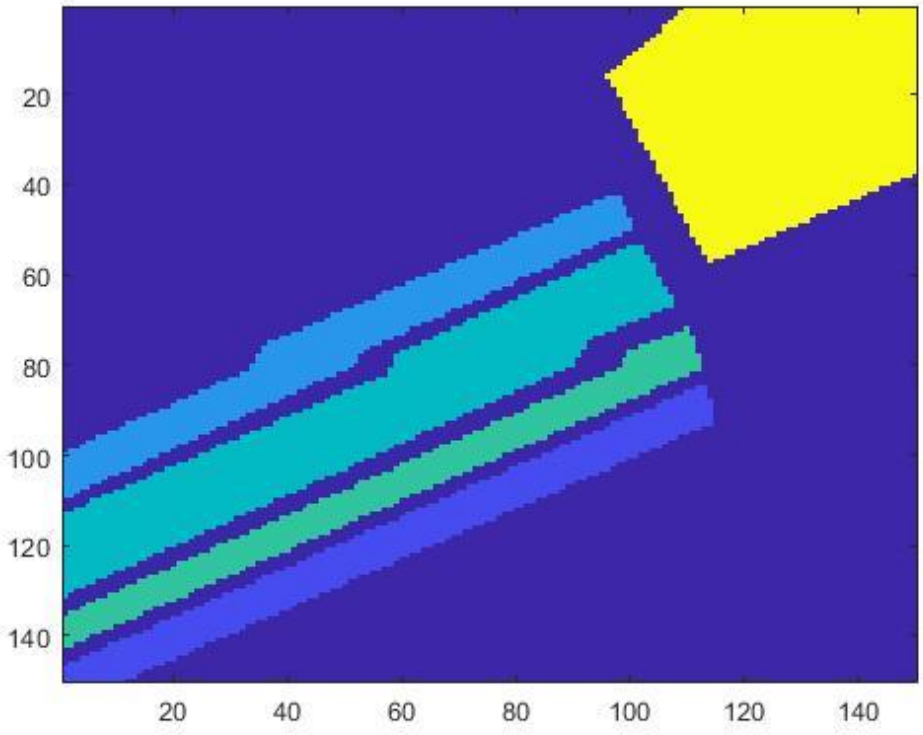
Όπως φαίνεται, το ποσοστό επιτυχίας του αλγορίθμου αυτού είναι αρκετά καλό αφού έκανε misclassify μονάχα 18 στοιχεία. Βέβαια το γεγονός ότι παίρνουμε μονάχα τον πρώτο πιο κοντινό γείτονα μπορεί να οδηγήσει σε λανθασμένα αποτελέσματα. Το πρόβλημα αυτό έρχεται να φτιάξει ο αλγόριθμος κοντινότερων γειτόνων, που ταξινομεί τα δεδομένα βάσει την κλάση που έχουν οι περισσότεροι γείτονές τους. Επιπλέον, τα αποτελέσματα των αλγορίθμων φαίνονται καλύτερα στα operation sets τα οποία φαίνονται στην οπτικοποίηση στο τέλος του αρχείου αυτού.

k-Nearest Neighbors

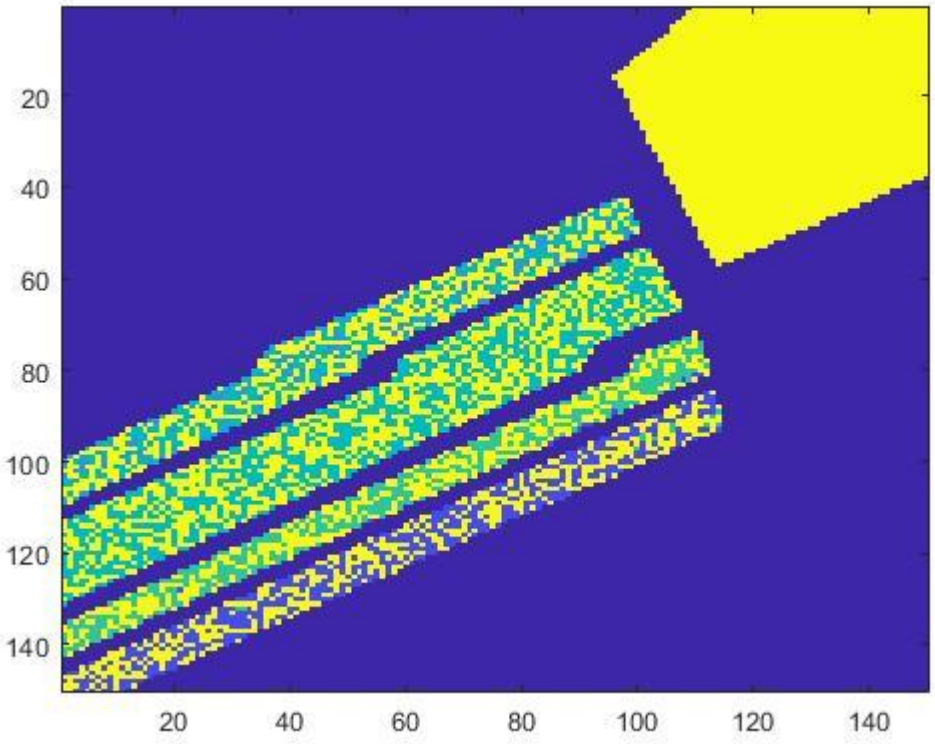
Τον αλγόριθμο κ-κοντινότερων γειτόνων, αρχικά τον υλοποίησα με grid-search για K= [3 5 7 9 11 13 15 17] και cross validation. Σπάμε τα sets σε cross-validation sets και για κάθε ένα υλοποιούμε τον αλγόριθμο knn με αυτό το set ως train και τα υπόλοιπα ως test sets. Τέλος, αφού πάρουμε τις προβλέψεις κάθε CV-set, υπολογίζω το accuracy αυτού του ταξινομητή και βρίσκω τον καλύτερο εξ' αυτών. Ο καλύτερος ταξινομητής ήταν αυτός των 3 κοντινότερων γειτόνων που έδωσε **max_accuracy = 0.9861.** Τον αλγόριθμο τον υλοποίησα με δέντρο στο οποίο τοποθέτησα το σύνολο Εκπαίδευσης με την βοήθεια του KDTreeSearcher.

Για την οπτικοποίηση των δεδομένων έχουμε

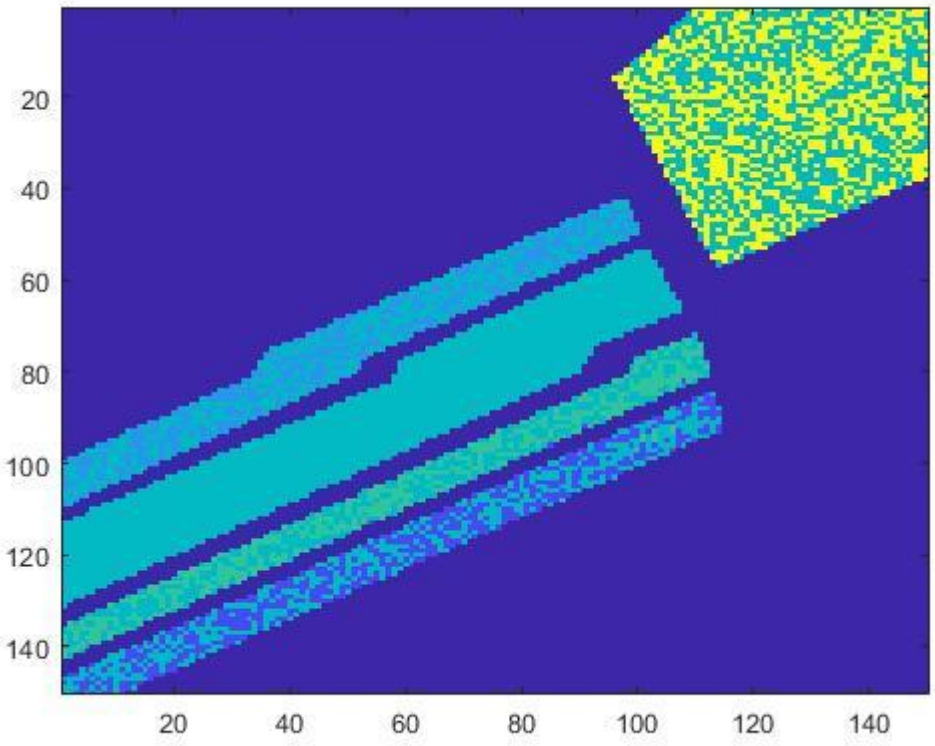
Αυθεντικά Δεδομένα



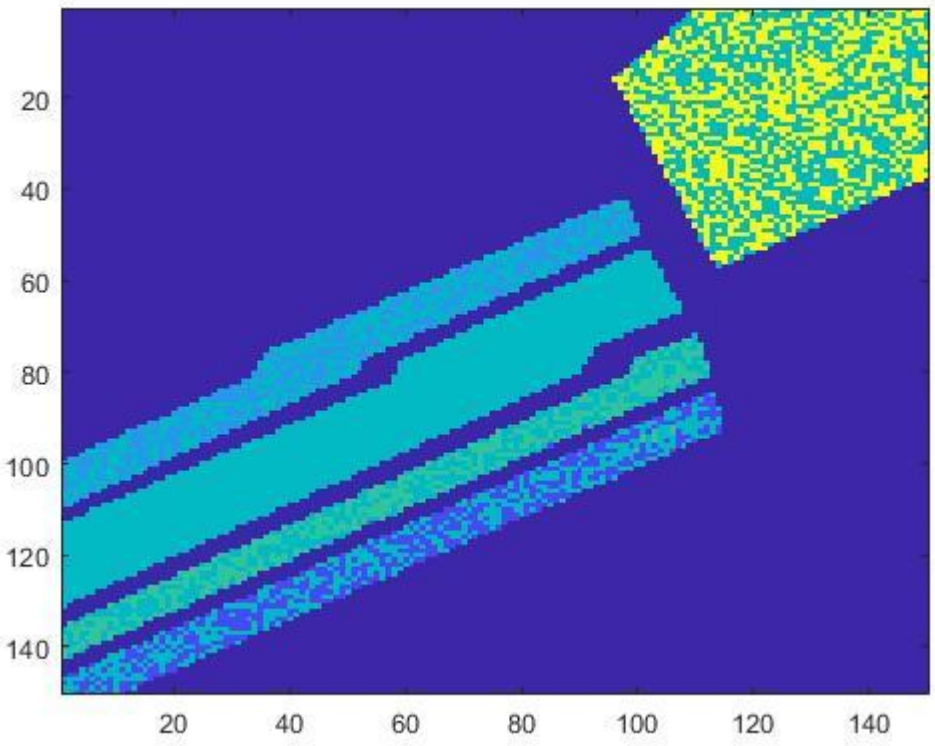
Naive Bayes Prediction



Euclidean Distance Predictions



3-nearest Neighbors



ΠΑΡΑΡΤΗΜΑ ΚΩΔΙΚΑ

Ο κώδικας που χρησιμοποιήθηκε υπάρχει και στο αρχείο project_m_file.m

Παρακάτω δίνεται ο κώδικας για εφαρμογές στο Train_set, με τα υπόλοιπα να είναι παρόμοια.

Naïve-Bayes

```
% Calculate real a priori
```

```
for i = 1:5
    Probabilities(i) =
sum(Train_array_response(i))/length(Train_array_response);
end
% Calculate mean and standard deviation

for i = 1:5
    mu(i,:) = mean(Train_array((Train_array_response==i),:),1);
    sigma(i,:) = std(Train_array((Train_array_response==i),:),1);
end

% Calculate likelihood and a posteriori
for i = 1:length(Train_array)

    p = normcdf(Train_array( i, :), mu, sigma);
    P(i,:) = Probabilities.*prod(p,2)';
end

% get predicted output for train set
[pv0,id]=max(P,[],2);
for i=1:length(id)
    train_predictions(i,1) = id(i);
end
```

Euclidean Distance

```
%Get mean of each class
for i=1:5
    mu( i, :) = mean(Train_array((Train_array_response==i),:));
end

dist = [];

train_predictions = [];
for i = 1:length(Train_array)
  for j = 1:5
    dist(i, j) = [sqrt(sum((Train_array(i, :) - mu(j, :)).^ 2 ) )];
  end
end
[dist, train_predictions] = min(dist,[],2);
```

Nearest Neighbors grid-cross_validation

```
for k = [3 5 7 9 11 13 15 17]

    consistency = 0;
    for i = 0:(cv-1)
        start_line = 1 + (i*no_of_features);
        end_line = start_line + no_of_features-1;


        X_test = Train_array(start_line : end_line, :);
        y_test = Train_array_response(start_line:end_line);

        X_train = Train_array(1:start_line,:);
        X_train = [ X_train; Train_array(end_line:len,:)];
```

```
        y_train = Train_array_response(1:start_line);
        y_train = [ y_train; Train_array_response(end_line:len)];

        size(X_train);
        size(y_train);


        tree = KDTreeSearcher(X_train, 'Distance' , 'euclidean' ,
'BucketSize',10);
        ids = knnsearch(tree, X_test, 'K',5);

        classes = ones(length(ids),5);
        for i = 1:length(classes)
            for j =1:5
                classes(i,j) = y_train(ids(i));
            end
        end

        predictions = zeros(length(classes),1);
        for i  = 1:length(classes)
            predictions(i) = mode(classes(i,:));
        end

        consistency = consistency + sum(predictions ==
y_test)/length(y_test)

    end

    accuracy = consistency/cv;
    if (accuracy > max_accuracy)
        max_accuracy = accuracy;
        best_predictor = k;
    end
end

best_predictor
max_accuracy
```

Confusion Matrix

```
confusion_matrix = zeros(5, 5);
for i=1: length(predictions)
     confusion_matrix(Test_array_response(i), predictions(i) ) =
confusion_matrix(Test_array_response(i), predictions(i) ) + 1;
end
fprintf("My KNN\n");
confusion_matrix
success_rate = trace(confusion_matrix)/  sum(sum(confusion_matrix))
```

Data Visualization

```
my_colours = zeros(150);
for i=1:length(Train_array)
    if (Train_array_response(i) == 1)
        my_colours(Train_array_pos(i,1), Train_array_pos(i, 2)) = 830;
    elseif (Train_array_response(i) == 2)
        my_colours(Train_array_pos(i,1), Train_array_pos(i, 2)) = 280;
    elseif (Train_array_response(i) == 3)
```

```
        my_colours(Train_array_pos(i,1), Train_array_pos(i, 2)) = 400;
    elseif (Train_array_response(i) == 4)
        my_colours(Train_array_pos(i,1), Train_array_pos(i, 2)) = 460;
    elseif (Train_array_response(i) == 5)
        my_colours(Train_array_pos(i,1), Train_array_pos(i, 2)) = 110;
    end
end
for i=1:length(Test_array)
    if (predictions(i) == 1)
        my_colours(Test_array_pos(i,1), Test_array_pos(i, 2)) = 830;
    elseif (predictions(i) == 2)
        my_colours(Test_array_pos(i,1), Test_array_pos(i, 2)) = 280;
    elseif (predictions(i) == 3)
        my_colours(Test_array_pos(i,1), Test_array_pos(i, 2)) = 400;
    elseif (predictions(i) == 4)
        my_colours(Test_array_pos(i,1), Test_array_pos(i, 2)) = 460;
    elseif (predictions(i) == 5)
        my_colours(Test_array_pos(i,1), Test_array_pos(i, 2)) = 110;
    end
end
for i=1:length(Op_array)
    if (op_predictions(i) == 1)
        my_colours(Op_array_pos(i,1), Op_array_pos(i, 2)) = 830;
    elseif (op_predictions(i) == 2)
        my_colours(Op_array_pos(i,1), Op_array_pos(i, 2)) = 280;
    elseif (op_predictions(i) == 3)
        my_colours(Op_array_pos(i,1), Op_array_pos(i, 2)) = 400;
    elseif (op_predictions(i) == 4)
        my_colours(Op_array_pos(i,1), Op_array_pos(i, 2)) = 460;
    elseif (op_predictions(i) == 5)
        my_colours(Op_array_pos(i,1), Op_array_pos(i, 2)) = 110;
    end
end
```