The power of teamwork

Increasing the performance of k-nearest neighbors using ensemble learning

Andrea Nodari - 466165

Abstract

Recently models are becoming more complex and accurate in predicting different types of data. However simple methods such as logistic regression and k-nearest neighbors still remains widely used and studied. In order to further increase the performance, researchers are studying a different approach than using more sophisticated models: combining different classifiers in order to achieve better results. These methods called ensemble learning are gaining increasingly popularity and importance. This report investigates how traditional approaches are able to solve classification problems and it evaluates how ensemble a learning method further imrpoves the performance.

I. Introduction

This report shows how using simple models and combining them is possible to reach high performance in classification tasks.

The classification task used in the experiments is composed by two smaller tasks: first,a binary classification of type of wines (i.e. red or white), second, the classification of quality of wines (i.e. assign to each example a number ranging from 1 to 7). The dataset contains 5000 examples with by 11 features: fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, and alcohol.

In order to introduce the two problems we propose a 3-dimensional visualization of the data (Figures 1 and 2).

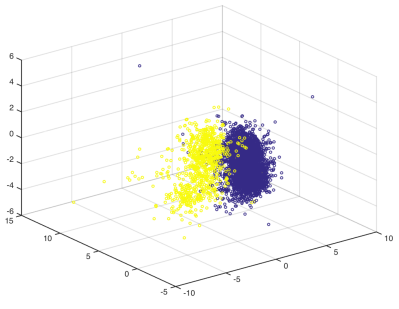

Figure 1: *Visualization of the type of wines*

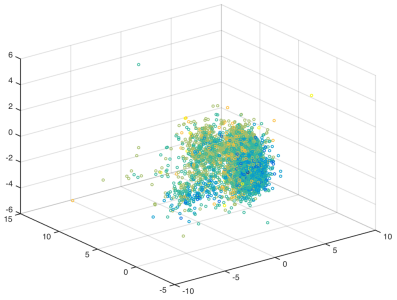

Figure 2: *Visualization of the quality of wines*

Since the data is originally is 11-dimensional, it has been projected using the Principal Component Analysis (Section IV.2) into a 3-dimensional space for visualization purposes. The figures show how in the classification of types case it easier to separate the two classes. Conversely, the classification of the qualities is a more challenging task.

The focus on this report is mainly on solving the problem of predicting the quality of wines, since the task is more challenging and interesting. Regarding the problem of predicting the types of wine, logistic regression has been used with good results (Section IV).

The following paragraphs propose an overview of the designed experiments regarding the second classification task. For further details about the methods, the experiments de-

sign, and the results see the respective sections.

The main question of this report is to understand if simple models can provide good result and what is the extent to which ensemble learning can further increase the performance.

Researches studied how simple methods, such as k-nearest neighbours, can be effective in critical tasks [10]. Furthermore several studies showed that combining different "base classifiers" in a proper way may result in improved performance [11].

The approaches used in the second task are the following. First, we use logistic regression with one-vs-rest technique, the results in this case are poor. Other approaches have been tried, such as feature selection and feature extraction. The performance in this case slightly improve, however the results are still not satisfactory.

These first experiments lead to a change of the model: going from a parametric method, the logistic regression, to a non-parametric method: k-nearest neighbors (k-NN).

Despite the simplicity of k-NN, it works remarkably well in this context. The reason of that can be found in the data itself: if a wine expert rate a wine with a high grade, given another wine with similar chemical characteristics it should have the same taste and be assigned to a high grade as well. Some of the ways tried to tune this model include: changing the type of distance and changing the value of k.

One of the main goal of this paper is to show that the performance can be further increased by utilizing ensemble learning. In the last experiment we propose some methods to combine k-NN classifiers to achieve better performance. The different methods include generating different k-NN classifiers using random sub-sampling of the features, random subsampling of the dataset and a combination of both.

II. Methods

This section proposes an overview of the methods and techniques utilized in the experiments.

I. Classification Algorithms

I.1 Logistic Regression

Logistic regression is one of the most used classification algorithm. The hypothesis is the following:

$$g(x_{new}) = \frac{1}{1 + e^{\theta^T x}}$$

The parameters θ are found by minimizing this function:

$$Cost(\theta) = \Big(-\frac{1}{N} \sum_{t=1}^{N} t^t log(y^t) + (1 - r^t) log(1 - y^t) \Big) + \\ + \frac{\lambda}{2N} \sum_{j=1}^{N} \theta_j^2$$

The second term of the equation is necessary for regularization, which is a technique to avoid overfitting. In particular is crucial to choose the right value for λ: the larger the value of lambda the simpler the model will be because the parameter will be forced to be small. Conversely a small value for λ allows more complex models. A correct value for lambda should be a trade-off.

Since the logistic regression algorithm can be used only for binary classification it is necessary to change the problem in order to use logistic regression for multi-class classification.

A possible approach is the one-vs-all strategy. The main idea of the algorithm is to train c models, where c is the number of classes. Each classifier is trained to predict if an example belongs to the given class or the rest. The final prediction is computed taking class that is predicted with the highest confidence among the c classifiers [5] .

I.2 k-Nearest-Neighbors

K-Nearest Neighbor is a non-parametric classification algorithm. Conversely to parametric methods in non-parametric algorithms it is not necessary to learn some parameters from the data. The classification is achieved by looking at the data itself.

K-Nearest Neighbors is an example of this type of algorithms. First of all it is necessary to choose a metric for the distance between to points: $d(x, x')$. Some example of distance metrics are Euclidean, Minkowski, Chebychev, and Cityblock (Appendix B).

Given the distance metric d it is possible to calculate the nearest neighbor of a point x:

$$(r_n, x_n) = argmin_{(r',x')\in X}\, d(x, x')$$

Namely, (r_n, x_n) is the point that minimize the value of distance, and r_n is the class of the nearest point. In the case of $k = 1$ the classifier predicts the class of the new point x as the same class of the nearest point, which in this case is exactly r_n.

In general it is possible to find the k-nearest neighbors. Therefore we compute from the nearest point to the k^{th} nearest point. Afterwards, once that all the k points have been computed the classifier assigns the class r to the new point, where r is the majority class of the classes of the k-points. If there is a tie, it is possible to pick the class of the closest point belonging to one of the classes as tie-breaker.

K-nearest neighbors algorithm is well studied and used algorithm, it is widely used even in crucial task such as risk assessment in credit cards [10].

II. Ensemble Methods

One of the main goals of this article is to show that ensemble method further enhance the performance. Ensemble methods are about combining different models in order to have an over better model. They differ from each other in how do the outputs of single models are combined, and how the different models are generated [6]. In this report we propose voting using the mode as combining function. We investigate bagging, random-forest and a combination of the two approaches to generate different models.

II.1 Voting

Voting is an approach to combine the outputs of different models in a single one. This strategy also called *multiexpert* combination methods [6]. The idea is to generate different models that work in parallel and combine the results using a function f. In other words, denoting the models with $d_1, d_2, ...d_n$ the final output will be:

$$y = f(d_1(x), d_2(x), ..., d_n(x))$$

In this paper we propose the mode of the outputs (i.e. majority voting) as combining function.

II.2 Bagging

Bagging, or *bootstrap aggregating* is a method to generate different models. The main idea is to draw from the dataset a certain number of examples and use this as a training set for a model. The difference of the various models is, therefore, left to chance [6]. In this report we investigate the performance of combining 1-NN classifiers a random subset of the training set [7].

II.3 Random Forest

Random Forest are an ensemble learning method that consists in combining decision trees classifiers [8]. This methods usually consist in random selection of the features to split. In this report we use 1-NN as base model and we exploit idea of randomly pick a set of features to obtain diverse classifiers.

III. Evaluation approaches

III.1 k-Fold Cross-Validation

K-Fold Cross-Validation is a model validation technique widely used in machine learning. Given a dataset of examples, we split it in "k" parts or folds. Afterwards, for each fold, a model is trained using "k-1" folds and tested on the remainder of the dataset. Finally the

average of the errors is computed. This approach useful to have a better estimation of the validation error of a given model. Algorithm 1 formally shows the steps of this technique.

Algorithm 1 K-Fold Cross-Validation

Input:
$X = (r^t, x^t)_{t=1}^N$
$k \in N$, a classification algorithm

Output:
ϵ validation error

Split X in k folds X_i
for $i \in 1...k$ **do**
    Train the model using $X - X_i$
    Compute ϵ_i: the error of the model on X_i
$\epsilon \leftarrow \sum_{i=1}^{k} |X_i|/|X| * \epsilon_i$
return ϵ

III.2 F-Score and Accuracy

In order to evaluate the models it is necessary to use a metric of the performance. In this report we used F-Score as measure of the performance.

$$F_{score} = 2\frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

$$precision = \frac{\text{true positive}}{\text{true positive} + \text{false positive}}$$

$$recall = \frac{\text{true positive}}{\text{true positive} + \text{false negative}}$$

In some cases we also describe the performance with the accuracy:

$$Accuracy = \frac{\text{true positive} + \text{true negative}}{N}$$

IV. Feature transformations

IV.1 Features Selection

Feature Selection is a technique to select a subset of important features ignoring the remainder. The main idea is that some features may be redundant, too noisy, or not useful for the classification, therefore ignoring them may result in better models.

The are two approaches to compute feature selection: exhaustive search and forward/backward search. The former may achieves better result, but it is computationally expansive ($\mathcal{O}(2^n)$ where n is the number of features) because it checks all possible combinations of the features and picks the best one. The latter is a greedy approach: it is faster, however it computes an approximate solution.

Algorithm 2 shows how to compute features selection using exhaustive search, while algorithm 3 shows forward search.

Algorithm 2 Exhaustive Features Selection

Input:
$X = (r^t, x^t)_{t=1}^N$
Output:
Features
(subset of features with the lowest error)
$n \leftarrow$ number of features
$E_{current} = \infty$
$F_{current} = \emptyset$
for each possible combination of n features **do**
    $F \leftarrow$ current combination
    Train the model using F features
    Compute ϵ: the error of the model using F as features
    if $E_{current} > error$ **then**
        $F_{current} \leftarrow F$
        $E_{current} \leftarrow E$
return $F_{current}$

Algorithm 3 Forward search

Input:
$X = (r^t, x^t)_{t=1}^N$
Output:
Features
(subset of features with the lowest error)
$n \leftarrow$ number of features
$E_{current} = \infty$
$F_{current} = \emptyset$
for for each feature f **do**
    $F \leftarrow F \cup f$
    Train the model using F features
    Compute ϵ: the error of the model using F as features
    if $E_{current} > error$ **then**
        $F_{current} \leftarrow F$
    else
        return $F_{current}$
return $F_{current}$

IV.2 Features Extraction: Principal Component Analysis

Feature extraction is a more general technique to achieve the same goal as feature selection (Section IV.1). The main idea is to project the data in a lower dimensional space so that we keep only the most informative features. One of the most important algorithm is Principal Component Analysis (PCA).

Algorithm 4 shows more formally the steps of the algorithm.

Algorithm 4 Principal Component Analysis

Input:
$X = (x^t)_{t=1}^N, x^t \in \mathcal{R}^d$
k, the number of dimension to project the data ($k < d$)
Output:
$Z = (z^t)_{z=1}^N, z^t \in \mathcal{R}^k$

$X' \leftarrow$ center-data(X)
$S \leftarrow$ covariance-matrix(Y)
$Eigenvectors, Eigenvalues = eig(S)$
$U_{reduced} \leftarrow$ k eigenvectors corresponding with the k largest eigenvalues
return $Y * U_{reduced}$

III. Experiments

In the following sections are presented the final setup for the two experiments: predicting the types of wine and predicting the qualities of wine.

I. Predicting the types of wine

In order to accurately predict the type of wines logistic regression has been used. There are no preprocessing steps for the data, since features manipulation technique such as PCA or feature selection were not successful.

The crucial parameter to tune in the proposed logistic regression (Section I.1) is the regularization term λ.

In order to investigate the best parameter the following we followed the following steps. A preliminary step consists in splitting the dataset into *training set*, *validation set* and *testing set*.

- The training set is used to train models with different values for λ.
- The validation set is used to compare the perfomance of the different models and to pick the best value for λ.
- The testing set is used to estimate the generalization error, namely how does the model perform on unknown data.

The metric used to compare the different models is F-Score.

An important remark is that once the best λ is chosen it is possible to merge the training set and the validation set to have better parameters.

The training of the logistic regression consists in minimizing the respective cost function. In order to effectively fit the parameters θ we used the function *fminunc* from the octave optimization library [2].

II. Predicting the qualities of wine

K-Nearest Neighbors is the model used for predicting the qualities of wine. In this case it is necessary to choose the right value for the parameter k, that is the number of neighbors to look in order to decide the predicted class. It is also necessary to investigate different types of distance metric in order to choose the one that maximize the performance.

In order to find the best value for the parameter k, 10 fold cross-validation has been used. The cross-validation technique provides as a result a validation error similar to the one calculated using the *validation set*, however the results are more robust. The approach is to try different values of k, ranging from 1 to 60, calculate the cross validation error and choose the k that minimize it.

In a similar way the distance metric has been chosen. The performance of k-NN classifier might change according to the chose distance metric, therefore it is important to try different ones. The following distances have been investigated:

- Euclidean
- Minkowski with exponent 3
- Chebychev
- Cityblock

For the formulas of different distance metrics refer to Appendix B.

It is necessary to carefully choose also the strategy to tie a break. In case $k > 1$, in fact, it is possible that two classes have the same number of elements. Some strategy include choose one class random among the competing classes, choose the class of the closest among the competing classes, choose the smallest.

In the experiment we decided to choose the class corresponding to the closest class. Figure 3 illustrates the strategy.

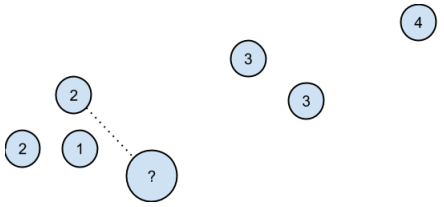

Figure 3: *Using 5-nearest neighbors result in a tie of classes 2 and 3. The new point will get class* 2 *because among the points belonging to* 2 *and* 3, 2 *is the closest*

Finally we designed the experiment with ensemble learning, the design is the following.

Three experiments are designed to test the performance of ensemble learning, all of the utilize k-NN classifiers (using the k that gives the maximum performance) and majority voting, however they differ in the way the different classifiers are generated. The first experiment generate different classifiers using different subset of the training set, the second using different subset of the features, the third combining the two previous strategy.

IV. Results

I. Predicting the types of wine

Logistic regression leads to high accuracy. Table 1 shows some details of the result: training of the model takes 4.7 seconds, moreover accuracy and f-score are high.

Total CPU time for training	4.7 s
Selected λ	0.4
Accuracy on test set	0.991
F-Score on test set	0.97

Table 1: *Results from the experiment of predicting the types of wine.*

II. Predicting the Quality of Wines

Using the logistic regression with one-vs-rest strategy result in unsatisfactory result in this task (Table 2).

Total CPU time for training	66.8 s
Selected λ	8.6
Accuracy on test set	48.2

Table 2: *Results from the experiment of predicting the qualities of wine using logistic regression with one-vs-rest model.*

Using k-nearest neighbors, despite the simplicity of the model result in better result. Figure 4, 5, 6, and 7 show the accuracy of the model using different value for k with different distance metrics.

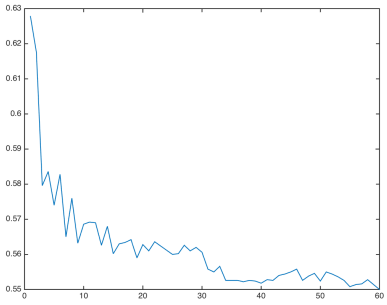

Figure 4: *Accuracies of k-NN using Euclidean distance with different k*

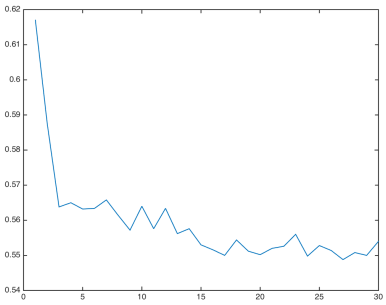

Figure 5: *Accuracies of k-NN using Minkowski distance with different k*

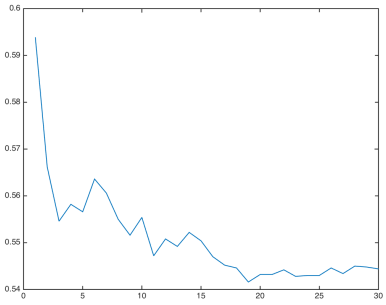

Figure 6: *Accuracies of k-NN using Chebychev distance with different k*

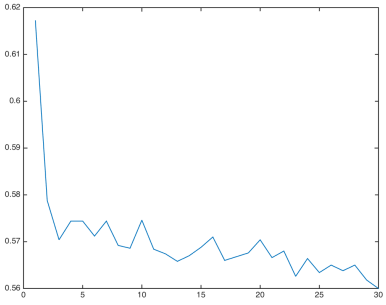

Figure 7: *Accuracies of k-NN using cityblock distance with different k*

The figures depict a similar pattern using different distance measures. In every case, in fact, the best classifier is the one with $k = 1$.

The Euclidean distance outperforms the

other distance measures resulting in an accuracy of 62.5%.

The F-Score using Euclidean distance and $k = 1$ is showed in Table 3.

In order to understand if ensemble learning improves the performance, we propose a comparison between three techniques to generate different 1-NN classifiers: using a random subset of the training set for each model, using a random subset of the features for each model, and using a combination between the two methods.

In each case we generate different models and using the mode of the output of the single models as final outputs.

Figures 8, 9, and 10 show how the accuracies change according to the number of models generated. The experiment has been repeated multiple time and the result are depicted using boxplot. Boxplot is a visual representation of the results such that "on each box, the central mark is the median, the edges of the box are the 25th and 75th percentiles, the whiskers extend to the most extreme data points not considered outliers, and outliers are plotted individually " [1]. Therefore, it is possible to understand also how the accuracies varies in the different experiments.

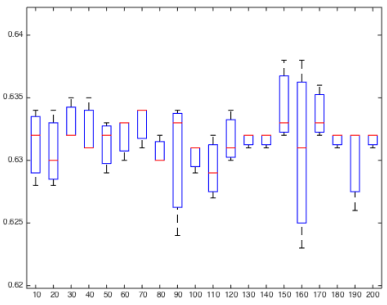

Figure 8: *Accuracies of ensemble learning using random subset of the training set, varying the number of generated models*

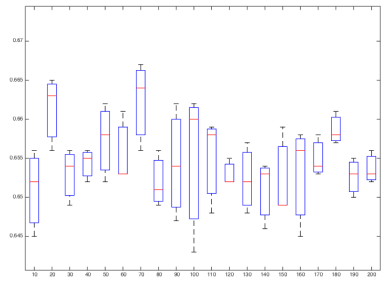

Figure 9: *Accuracies of ensemble learning using random subset of the features, varying the number of generated models*

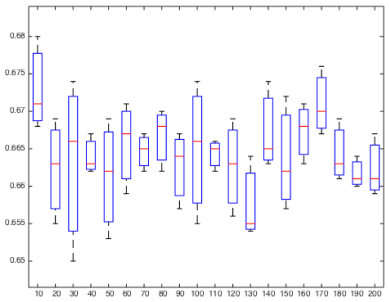

Figure 10: *Accuracies of ensemble learning using both random subset of the features and random sampling of the training set, varying the number of generated models*

Total Score	Q1	Q2	Q3	Q4	Q5	Q6	Q7
0.81132	0.00	0.46	0.63	0.64	0.67	0.31	0.00

Table 3: *Total F-Score and F-Score for each class using 1-NN classifier with euclidean distance.*

Table 4 shows a comparison between the best performance of the different methods

in the previous experiments. Both the three ensemble methods provide improved results, however using random features sub-sampling or both strategies seem to have the best result improving the performance of a single 1-NN classifier by 4%.

V. Conclusion

The first experiment shows that logistic regression performs well in the task of classifying the types of wine. In this task the two classes are linearly separable and logistic regression algorithm produces a very accurate model. Even from the computational side logistic regression is good: the training phase takes only few seconds.

However in the task of predicting the qualities of wine the same algorithm is not satisfactory. This task is more challenging and a different approach is needed.

In this paper we propose a simple non-parametric method: k-nearest neighbors. Despite the simplicity of the method, 1-NN works remarkably well.

The reason why it works may be found in the data itself. The dataset human labeled according to taste. It is therefore sensible to assume that wines with the same chemical characteristics might have the same quality.

A goal of this paper is understand if using the same model with ensemble methods may result in better performance. In order to validate this statement three methods of generating different classifiers have been tested: random sub-sampling of the training set (bagging), random sub-sampling of the features (random forest), and the combination between the two approaches.

The results show that combining different classifiers can increase the performance. We obtained an accuracy improvement from 1% to 4%.

Some possible future works may be to include a comparison between other ensemble learning methods, such as Adaboost [13, 12], or *stacking* [6]. Researchers also studied combination of trees: researchers may investigate further about the performance of the classification using random forest using decision trees as base classifier [8]. Other possibilities to improve the results are to measure the performance with different model, for instance support vector machine (SVM) [9].

Method	Number of models	Accuracy (median)
Subset of training set	70	0.63
Subset of the feature	70	0.66
Both	10	0.67

Table 4: *Results of the experiments with ensemble learning*

References

[1] boxplot references. `http://se.mathworks.com/help/stats/boxplot.html`. Accessed: 2014-11-18.

[2] fminunc function. `https://www.gnu.org/software/octave/doc/interpreter/Minimizers.html#XREFfminunc`. Accessed: 2014-11-18.

[3] Matlab details for k-nn classifier. `http://se.mathworks.com/help/stats/classificationknn-class.html`. Accessed: 2014-11-18.

[4] Octave cpu library for measuring performance. `https://www.gnu.org/software/octave/doc/interpreter/Timing-Utilities.html#XREFcputime`. Accessed: 2014-11-18.

[5] Online course: Machine learning by prof. andrew ng. `https://www.coursera.org/course/ml`. Accessed: 2014-11-18.

[6] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.

[7] Leo Breiman. Bagging predictors. *Mach. Learn.*, 24(2):123–140, August 1996.

[8] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001.

[9] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[10] W. E. Henley and D. J. Hand. A k-nearest-neighbour classifier for assessing consumer credit risk. 1996.

[11] Richard Maclin and David Opitz. Popular ensemble methods: An empirical study. *arXiv preprint arXiv:1106.0257*, 2011.

[12] Robert E Schapire. A brief introduction to boosting. In *Ijcai*, volume 99, pages 1401–1406, 1999.

[13] Ji Zhu, Hui Zou, Saharon Rosset, and Trevor Hastie. Multi-class adaboost. *Statistics and Its*, 2009.

A. Code Availability

Part of the code used in the experiments of this report is available at `https://github.com/nodo/machine-learning-challenge`. In the following sections we propose only some interesting excerpts from the code. For further details of the implementation of logistic regression, k-NN, PCA, features selection, refer to the link.

B. Distance Metrics

Euclidean	$d(p,q) = \sqrt{\sum_{i=1}^{N}(p_i - q_i)^2}$
Minkowski with exponent 3	$d(p,q) = (\sum_{i=1}^{N} \lvert p_i - q_i \rvert^3)^{1/3}$
Chebychev	$d(p,q) = max_i(\lvert p_i - q_i \rvert)$
Cityblock	$d(p,q) = \sum_{i=1}^{N} \lvert p_i - q_i \rvert$

Table 5: *Formulas of different distance metrics used in the k-nearest neighbors classifier.*

C. K-Nearest Neighbors

In this section we propose two excerpts of code that predict the class of a new point using 1-NN with Euclidean distance. Section I shows a naive implementation, while in Section II it is possible to see an improved version. The performance are drastically improved in the second case (about 100%), as it is possible to see in Table 6. The comparison shows the CPU time using $k = 1$.

The performance improvement is mainly due to the vectorized implementation that exploit built-in octave optimization. A further improvement in performance may be to use kd-trees [3].

The CPU has been measure using the octave library [4].

Implementation	CPU time
Naive	0.1036 s
Vectorized	0.0013 s

Table 6: *Comparison of the performance of k-NN implementation*

I. Naive implementation

```
% Return the class of the closest point
function class = findClass(X, p)
  m_distance = intmax;

  idx = -1;
  for i=1:rows(X)
    d = distance(X(i,1:end-1), p);
    if (d < m_distance)
      m_distance = d;
```

```
      idx = i;
    end
  end

  class = X(idx, end);
end

% Compute euclidean distance
function d = distance(x1, x2)
  d = norm( x1 - x2 , 2 );
end
```

II. Fast implementation

```
% Return the class of the closest point
function class = findClassFast(X, p)

  % vectorized implementation of euclidean distance
  for i=1:length(p)
    X(:, i) = X(:, i) - p(i);
  end
  N = sqrt(sum(abs(X(:, 1:end-1)).^2,2));

  [ordered, index] = sort(N);

  % create a counter for each class
  num_classes = 7
  counters = zeros(num_classes,1);
  for i=1:num_classes
    counters(X(index(i), end)) += 1;
   end

  [sorted_counters, index_counters] = sort(counters, 'descend');
  % if there is no tie, just return the closest
  if (sorted_counters(1) > sorted_counters(2))
     class = index_counters(1);
  else
    % random tie-breaking strategy
    maj = majority(counters);
    class = maj(randperm(length(maj)))(1);
  end
end
```

D. Ensemble learning

In this section we propose the code used to test the ensemble learning methods. In this case both sub-sampling of the features and the training set is used.

This code is used to generate the boxplot, therefore it tries the experiment multiple times changing the number of generated models and measuring the performance.

```
% load the data
data      = load('../../../data/data.csv');
test_data = load('../../../data/test.csv');
train  = data(:, 1:end-2);
labels = data(:, end-1);
test      = test_data(:, 1:end-2);
qualities = test_data(:, end-1);
rows      = size(test, 1);

% number of maximum models
NUMBER_OF_MODELS = 200;
probes = NUMBER_OF_MODELS / 10;

% number of experiments
NUMBER_OF_EXPERIMENTS = 10;
accuracies = zeros(probes, TRY);

for t = 1:NUMBER_OF_EXPERIMENTS
    for m = 1:probes
        ensemble = zeros(0,0,0);
        for model_id = 1:NUMBER_OF_MODELS
            k = 1;

            % select random features
            number_of_features = randi([1, size(test, 2)]);
            rand_perm = randperm(size(test,2));
            features = rand_perm(1:number_of_features);

            % select random subset of training set
            num = randi([1, size(train,1)]);
            perm = randperm(size(train,1));
            training_sample = perm(1:num);

            train_m = train(training_sample, features);
            labels_m = labels(training_sample, :);
            test_m  = test(:, features);

            % train the model (some parts are omitted for readability)
            model = fitcknn(train_m, labels_m, 'NumNeighbors', k, ...)
            predictions = predict(model, test_m);
            ensemble(model_id, :, :) = predictions;
        end
        ensemble_predictions = zeros(size(test_data, 1),1);

        % create an array of predictions
```

```
        for i = 1:size(test_data, 1)
            % the prediction is the mode of the models predictions
            ensemble_predictions(i) = mode(ensemble(:, i, :));
        end

        % save accuracy
        accuracy = sum(ensemble_predictions == qualities)/rows;
        accuracies(m, t) = accuracy;
    end
end

% plot the boxplot
v = 1:probes;
v = v .* 10;
boxplot(accuracies', 'labels', v);
```