Analysis of ensemble techniques on tree-based classifiers and SVM for breast cancer classification

Giovanni Cioffi

277046

Mathematics in Machine Learning

M.Sc. in Data Science and Engineering, Politecnico di Torino

Contents

1 Introduction

In this project I am going to analyse some of the topics examined in "Mathematics in Machine Learning" course offered at Politecnico di Torino, M.Sc. in Data Science and Engineering. I will build a binary classification model for predicting whether a breast mass is malignant or benign based on Breast cancer Wisconsin (diagnostic) dataset. In particular, I will focus on the **bias-variance trade-off** and how **ensemble techniques** such as Bagging, Random Forest and Adaptive Boosting used on tree-based classifiers and Support Vector Machines can handle it. All the steps of the data pipeline will be explored, including data pre-processing, features selection, Principal Component Analisys and Singular Value Decomposition, algorithms' details and implementation and, finally, the results.

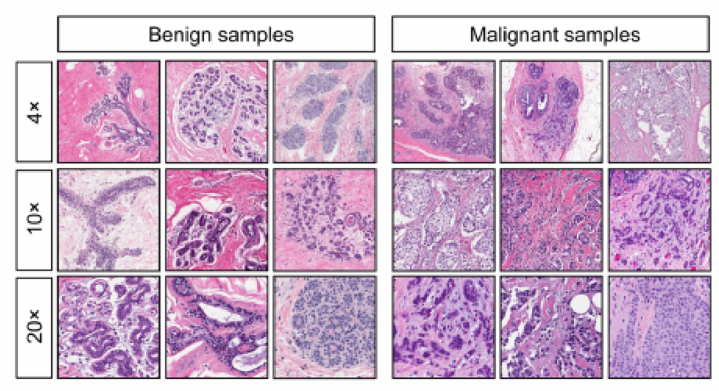

Figure 1: FNA digitalized images of breast masses

2 Data exploration and preprocessing

2.1 Dataset description

Breast cancer Wisconsin (diagnostic) dataset from University of Wisconsin is made of 569 instances each one describing the characteristics of the cell nuclei in the digitalized image of fine needle aspirate (FNA) of a breast mass. The target labels are WDBC-Malignant or WDBC-Benign depending on whether the breast mass resulted to be malignant or benign. The dataset presents 30 numeric predictive attributes obtained from 10 “base features”. A class label related to the diagnosis of the breast mass is associated with each record. The feature and target names are shown in Table 1:

Feature name	Feature type	Feature description
radius	Float64	mean of distances from center to points on the perimeter
texture	Float64	standard deviation of gray-scale values
perimeter	Float64	-
area	Float64	-
smoothness	Float64	local variation in radius lengths
compactness	Float64	$\frac{perimeter^2}{area-1.0}$
concavity	Float64	severity of concave portions of the contour
concave points	Float64	number of concave portions of the contour
symmetry	Float64	-
fractal dimension	Float64	“coastline approximation” - 1
diagnosis	**Int32**	**”benignant” or ”malignant”**

Table 1: Attributes description: base features and target labels.

For each “base feature”, the mean, standard error, and “worst” or largest (mean of the three worst/largest values) of these features were computed for each image, resulting in 30 features. Figure 2 bar chart shows that the dataset is unbalanced towards the WDBC-Benign class, with 357 against 212 entries. No missing values were detected.

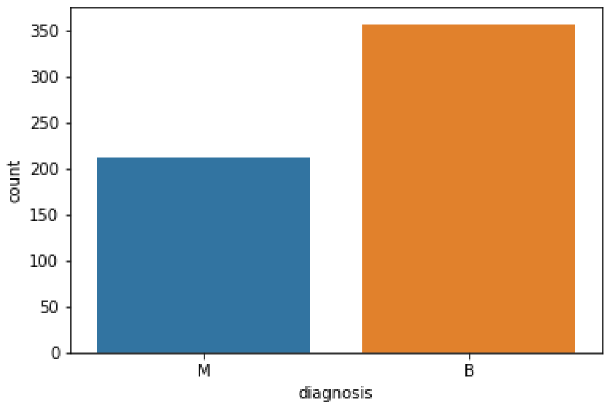

Figure 2: Barchart showing class unbalancing between *malignant* and *benign* samples

2.2 Feature selection: correlation and KDE

Correlation is any statistical relationship, whether causal or not, between two random variables. Essentially, it is the measure of how two or more variables are related to one another. **Pearson correlation coefficient** or linear correlation between two statistical variable X and Y, is the covariance of the two variables divided by the product of their standard deviations σ_X and σ_Y, the formula (for a population) is shown below:

$$\rho_{X,Y} = \frac{Cov(X,Y)}{\sigma_X \sigma_Y}$$

Pairwise Person coefficient was calculated in order to inspect correlation among features, the corresponding heatmap is shown in Figure 3.

The heatmap shows that many features are highly correlated (attributes presenting yellow squares), and this is probably due to the intrinsic characteristics of the dataset.

For the sake of visualization, some pair scatter plots of *mean texture*, *mean radius* and *mean area* are shown in Figure 4 in order to see the relationship between a high correlation and a linear pattern.

The more correlation coefficient is close to 1, the more scatter plots present an almost linear pattern. This is the case of *mean radius* and *mean area* with 0.98 Pearson correlation coefficient. Features pairs with correlation closer to 0 such as all the features combined with *mean texture* (about 0.32 correlation coefficient), instead, do not show a distinct linear pattern. Since correlated features may negatively affect classification models, for each couple presenting correlation coefficient higher than 0.90 in absolute value, I

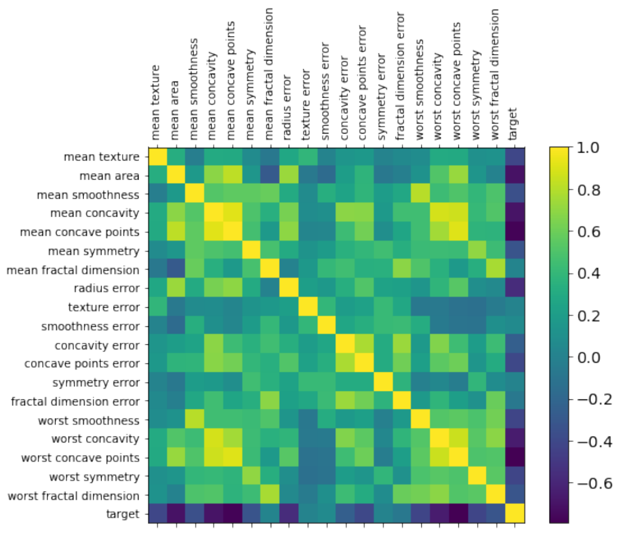

Figure 3: Heatmap showing pairwise Pearson correlation coefficient among couples of attributes

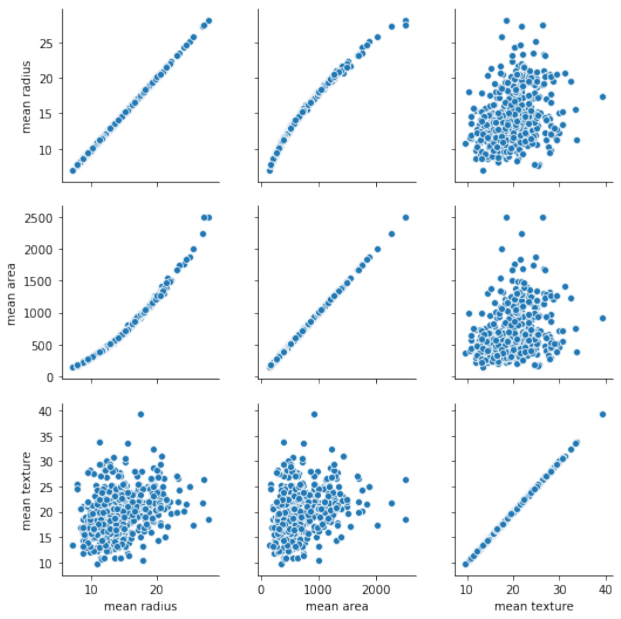

Figure 4: Distinction between correlated and uncorrelated features though pairwise scatter plot of *mean texture*, *mean radius* and *mean area* attributes showing or not showing linear patterns

decided to drop one of them. Finally, the following features were dropped: *mean perimeter, worst perimeter, perimeter error, mean radius, worst radius, area error, worst area and worst texture.*

In addiction, all features related to *compactness* where removed since this attribute is a function of *perimeter* and *area*.

To better visualize which features could be the most important in the classification phase, two of the most interesting *kernel density estimates* (KDE) plots are here reported. The KDE technique is a way to estimate the probability density function of a random variable through the empirical data distribution. For each feature and class label separately, KDE technique was applied in order to visually inspect which were the most "characterizing" features by inspecting the distribution of *malignant* versus *benignant* records of the same feature. In Figure 5 below, two KDE examples are shown.

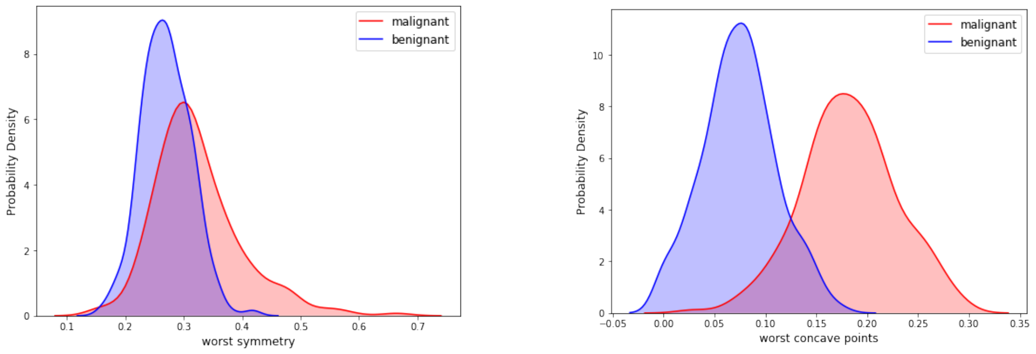

Figure 5: KDE plots for *worst symmetry* and *worst concave points*.

2.3 Data transformation: splitting normalization and outliers removal

Data were shuffled and split with stratification (in order to preserve the percentage of samples for each class) in training and testing set with a 7:3 ratio. The train set ended up having 250 malignant records and 148 benignant ones.

Since data are differently scaled there was need for **standardization**. Data were standardized by removing the mean of the training samples of that feature column and dividing by the standard deviation. Standardization is a common requirement for many machine learning estimators. They might behave badly if the individual features do not look like standard normally distributed data: Gaussian with zero mean and unit variance. The standard score of a sample x is calculated as follows

$$z = \frac{x - u}{s}$$

where u is the mean of the training samples and s is the standard deviation.

As mentioned in Data Description section, the dataset has no missing values. The only operation directly performed on the data was an **outlier detection** with boxplots and **deletion**. Boxplots are a standardized way of displaying the distribution of data based on the minimum $Q_1 - 1.5IQR$, first quartile Q_1, median, third quartile Q_3, and the maximum $Q_3 + 1.5IQR$. Boxplot visualization helped to detect outliers, they were initially filtered out using interquartile range method (IQR), that is a measure of statistical dispersion, being equal to the difference between $75th$ and $25th$ percentiles. Outliers were defined as observations that fall below the minimum or above the maximum, where $IQR = Q_3 - Q_1$. The following figure shows boxplots of all the standardized attributes:

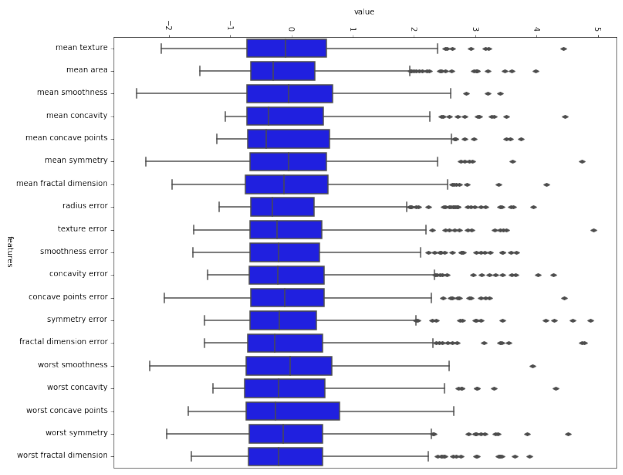

Figure 6: Boxplots of standardized attributes

IQR method for outliers deletion had some issues since it filtered out most of points belonging to *malignant* class ending up with a situation of enhanced strong unbalancing between classes. This would have worsened classification performances and pointed out that most of the outliers could be useful for the prediction. Thus, I decided to use an *interdecile* range in order not to remove too much samples. Only the training data were filtered out in order to not to tamper with the unknown data distribution of the targets. 11 malignant samples and 7 benign ones were finally dropped.

After outliers removals, data were re-normalized.

2.4 Dimensionality reduction: PCA and SVD

Principal component analysis linearly transforms data by simultaneously mapping them into a new space whose dimensionality is smaller and trying to keep most of the information of the original data. Dimensionality reduction can be done for several purposes: enhancing interpretability of data, finding meaningful structures of the data or for illustration purposes. For the purposes of this project, I am going to reduce the dimentionality in order to have a visual perspective on data trying to keep more information as possible.

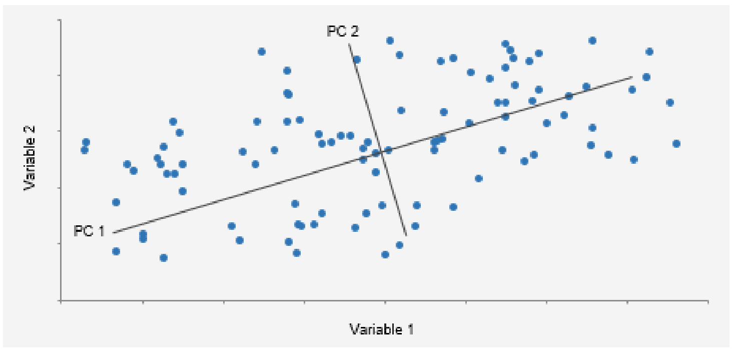

Figure 7: Principal components directions PC1 and PC2 in a 2D case

What PCA does is finding the "directions" of the data that explain most of the information present in them. Starting from our data matrix X of m samples and d dimensions (our attributes), PCA does the **eigendecomposition** of the covariance (between attributes) matrix $X^T X$ (d, d), where eigenvectors and eigenvalues show the direction and the magnitude of the spread of data respectively. We then obtain the matrix W of eigenvectors its columns representing the *principal components* are called *loadings*. The loadings are ordered depending on eigenvalues (and they are all uncorrelated), in the sense that the first column of W is associated with the highest eigenvalue and will explain more variance that the second, the third will explain more variance than the third and so on. Thus, we can choose the number r of components we want to keep by truncating W and taking the first r loadings. The matrix T of the new transformed data (or *scores*) will be the product between the original data matrix X and the truncated matrix W_r,

$$T_r = XW_r$$

of size (m, r), where r is the final number of dimensions we want to keep. It turns out that the computation of the covariance matrix and of its eigendecomposition is not necessary the most computationally efficient, here comes **Singular Value Decomposition**. We can decompose the original data matrix

$$X = U\Sigma V^T$$

where U and V^T respectively represent the left and right singular vectors and Σ is the diagonal matrix that has the singular values σ_i on its diagonal, where $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_m$. Since V is identical to the eigenvectors matrix W, it turns out that the truncated matrix of the scores can be obtained as

$$T_r = U_r \Sigma_r$$

Finding the number of components r is crucial. One can either decide to arbitrary decide a number of dimensions to take or to find that r that explains a certain amount of the variability of the data by putting a percentage threshold.

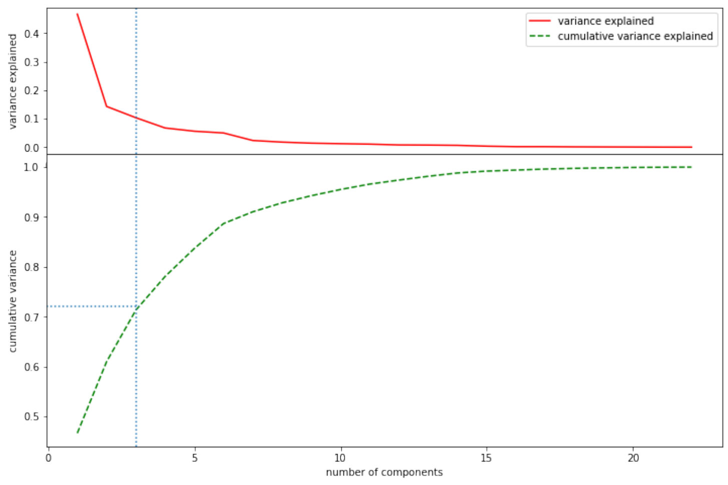

Figure 8: Cumulative variance and variance explained vs number of components. 3 components explain around 72% of the variance.

The plot in Figure 8 shows the number of components on the x-axis and the **cumulative variance** and **the variance explained** on the y-axis. As the number of components grows, the cumulative variance (equal to the cumulative normalized sum of the the singular values) of X grows until reaching a plateau while the variance explained by each of the component decrease since they are ordered (the first component explain most of the variance). A reduction to 3 components explains almost 72% of the variance.

After dimensionality reduction, data were plotted in 3 dimentions. The scatter plot shows that data are likely to be linearly separable. Thus, a Support Vector Machine algorithm should work properly (we will analyze its performances in Chapter 5).

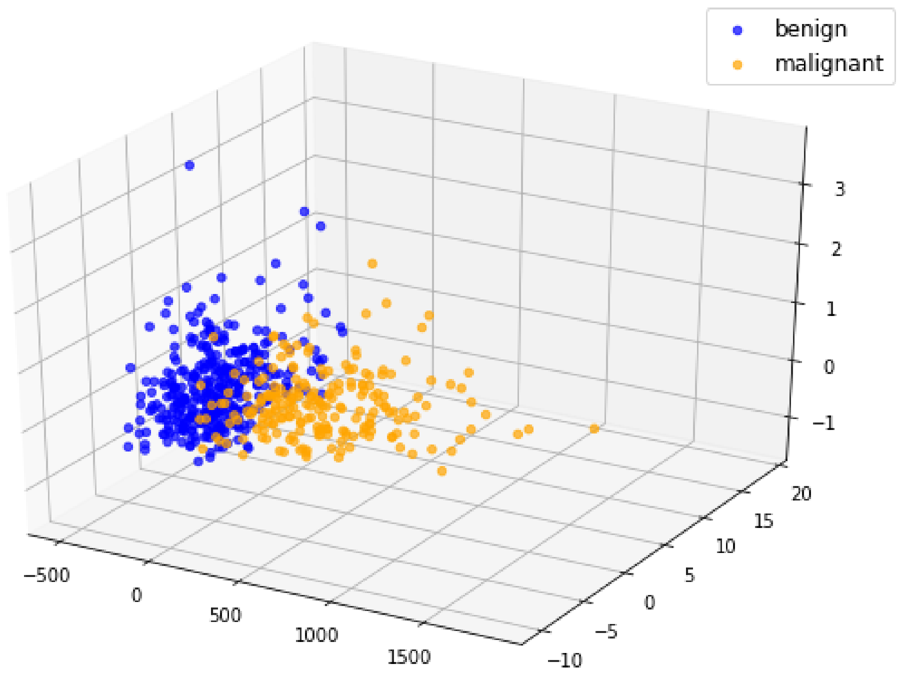

Figure 9: 3D scatter plot after PCA dimensionality reduction. Blue points represent benign records, orange points malignant.

3 Ensemble classification techniques

Ensemble classifiers use predictions made by several learning algorithms, usually belonging to the same family, to perform a classification task. Multiple models are trained to solve the same problem and combined to get better performances. The main hypothesis is that when *weak models* are correctly combined, we can obtain more accurate and/or robust models obtaining a prediction that is better than a prediction obtained with a single base model.

3.1 The bias-variance trade-off

Ensemble techniques are thought to tackle the *bias-variance trade-off*, introduced when learning through the *empirical risk minimization* (ERM, that is finding a predictor h that minimizes the empirical risk) restricted to a specific hypothesis class H, a set of predictors that reflects some prior knowledge about the task. It is the conflict in trying to simultaneously minimize the two following sources of error of an ERM algorithm over a class H:

- **Approximation error**, or *bias* of the algorithm towards choosing a hypothesis from H (it does not depend on the sample size), in other words the minimum risk achievable by a predictor in the hypothesis class.

- **Estimation error**, or *variance* of the algorithm that depends on the size or complexity of H. It results because the empirical risk (i.e. the training error) is only an estimate of the true risk, so the predictor minimizing the empirical risk is only an estimate of the predictor minimizing the true risk;

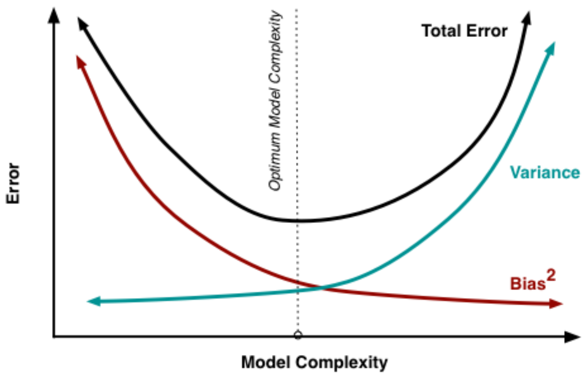

Figure 10: Bias-variance trade-off

High bias leads to **underfitting**, high variance leads to **overfitting**. Enlarging the hypothesis class can decrease the approximation error but at the same time can increase the estimation error as a rich (i.e. complex) H might lead to overfitting. On the other hand, choosing H to be a very small set might increase the approximation error and lead to underfitting.

One way of resolving the trade-off is to use mixture models and ensemble learning such as Bagging, Boosting or Random Forest.

3.2 Weak learnability vs strong learnability

A *weak learner* (or base model) is a learning algorithm lacking in complexity and easy to learn that, no matter what the distribution over the training data is, it will always perform better than chance when it tries to label data. In *strong learnability*, instead, the error is required to be arbitrarily small. More formally, under the PAC learning framework in a binary classification context, a learning algorithm is γ-weak learner for a class H if there exist a function m_H such that for every $\gamma \in (0, 1)$, for every distribution D over χ and for every labelling function $f : \chi \to \{\pm 1\}$, if when running the algorithm on $m > m_H$ i.i.d. examples generated by D, it returns a hypothesis h such that with probability of at least $1 - \gamma$, the true error

$$L_{D,f}(h) \leq 1/2 - \gamma$$

The lower the γ value, the weaker the learner, the more complex the ensemble technique should be to build an efficient strong learner. Most of the time, these basics models perform not so well by themselves either because they have a high bias or because they have too much variance to be robust. Thus, the idea of ensemble methods is to try reducing bias and/or variance of such weak learners by combining several of them together in order to create a strong learner (or ensemble model) that achieves better performances.

3.3 Bootstrap

Bootstrap is a resampling method that tackles the issue related to which data to use for training several learners in ensemble classification. Given a standard training set D of size m, when building k sources for the weak learner (usually having the same size m), one simply sample **with replacement** from the original dataset. Each of the new generated set will have roughly 64% of the unique samples of D, the others being duplicates (not deleted). All these samples can be considered almost independent from each other, their approximation of the true distribution being more accurate the larger

m is. In fact, fitting fully independent models would require too much data. So, we rely on the good "approximate properties" of bootstrap to fit models that are almost independent.

3.4 Bagging

The term stands for "**B**ootstrap **agg**regation", it is an ensemble technique that mainly focuses on **reducing the variability** of the simple model. It combines many weak models fitted on bootstrapped samples from the original dataset in an ensemble that has lower variance than the individual models.

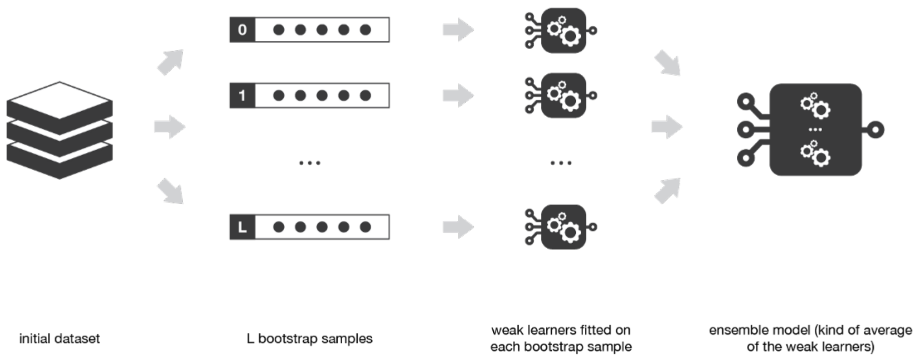
0
1
...
L
initial dataset
L bootstrap samples
weak learners fitted on each bootstrap sample
ensemble model (kind of average of the weak learners)

Figure 11: Bagging framework

The base models can be considered approximately independent and identically distributed (i.i.d.) as the bootstrap samples and, since different models are built independently, this process can be **parallelised**. The final prediction is taken by **majority voting** (for a classification task) with all contribution having the same importance. "Averaging" weak learners' outputs do not change the expected answer but reduce its variance just like averaging i.i.d. random variables preserve expected value but reduce variance. Thus, if we consider T weak learners, each one outputting a probability $p_i(x)$ for each class $i \in I$ (class set) for a query sample x, the ensemble prediction will be

$$f_{bag}(x) = \underset{i}{\operatorname{argmax}} \left\{ \frac{1}{T} \sum_{t=1}^{T} p_i^t(x) : i \in I \right\}$$

3.5 Boosting

The boosting paradigm allows the learner to have smooth control over bias-variance trade-off. The idea of boosting is to fit models **sequentially** (iteratively) such that the training of model at a given step depends on all the models fitted at the previous steps. Boosting produces an ensemble model that is generally **less biased** than the weak learners that compose it. For the purpose of this project I am going to use AdaBoost classifier.

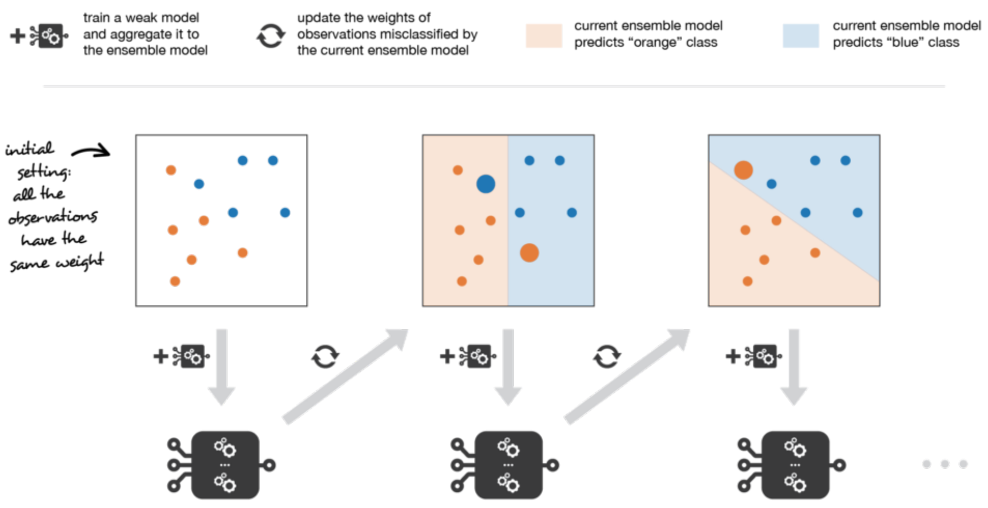
train a weak model and aggregate it to the ensemble model
update the weights of observations misclassified by the current ensemble model
current ensemble model predicts "orange" class
current ensemble model predicts "blue" class
initial setting: all the observations have the same weight

Figure 12: The Adaptive Boosting framework.

3.5.1 AdaBoost

AdaBoost (Adaptive Boosting) is an iterative algorithm composed of a sequence of consecutive rounds. At each step t, the algorithm defines a distribution over the training samples $S = (x_1, y_1), ..., (x_m, y_m)$ called D^t. Each weak learner gets as input the distribution and it is required to return an hypothesis h_t, whose error it is at most $0.5 - \gamma$, as previously defined. The final strong hypothesis h_s will be a weighted combination of weak ones. The adaptive policy introduced by AdaBoost is to assign a weight w_t to this weak hypothesis, whose magnitude is inversely proportional to the error ϵ and indicates how much this weak learner should be taken into account into the ensemble model. Using this weight, the last step is to update the probability distribution in such a way that the training samples on which h_t makes mistakes will have a higher probability mass. Since this will be the distribution input for the next learner, it introduces an enforcement to focus on the problematic samples.

A single step of AdaBoost algorithm at time t is described below:

1. given training samples S and their distribution D^t, fit the best possible weak model and return a weak hypothesis h_t;

2. compute the weight w_t associated with the returned hypothesis h_t depending on its error ϵ;

3. update the strong learner by adding the new hypothesis h_t multiplied by its weight w_t to the weighted sum;

4. update the distribution D^{t+1} that expresses which observations we would like to focus on at the next iteration;

At step 1, the algorithm simply trains a weak learner on the original data, each sample has the same probability mass $D^1 = (\frac{1}{m}, ..., \frac{1}{m})$. For each successive iteration, the distribution D is updated in a way that incorrectly predicted samples have their probability mass increased, whereas samples that where predicted correctly have their probability mass decreased. In this sense, AdaBoost is sequential because, as iterations proceed, examples that are difficult to predict receive increasing influence and each subsequent learner is forced to concentrate on the examples that were misclassified by the previous learner.

The main differences between bagging and boosting are the following:

- Bagging can be executed in parallel whereas boosting requires a sequential approach;

- Bagging considers all learners' contributions equal whereas boosting combines weighted contributions;

- Bagging mainly tackles the variability whereas boosting cares manly about bias (even if variance can also be reduced);

4 Application to tree and SVM based algorithms

My analisys will be focused on ensemble techniques applied on the following families of classification algorithms: 1**decision trees** and **SVMs**. I will briefly introduce their characteristics and discuss whether they can benefit from an ensemble approach (or not).

4.1 Decision Trees

Decision trees are predictors that assign a label associated with an instance by travelling from a root node to a leaf node in a tree. At each node of the root-to-leaf path, the successor child is chosen on the basis of a **splitting** (or partitioning) of the input space, which is segmented and stratified in several distinct and non-overlapping regions depending on the splits defined by the tree (as shown in the example of Figure 13).

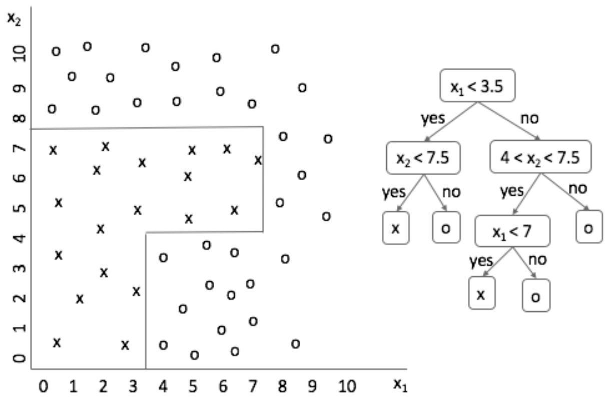

Figure 13: An example of decision tree (right) and the resulting partitioning (left) of the space in 2D case.

Each internal node represents an attribute of the dataset, each branch represents the different regions in which the attribute is split, and each leaf node represents a class label. Between all the possible ones, the splits are chosen based by exploiting some “gain” or “impurity measure”, such as *Gini Index*, that guarantee the goodness of the split (attributes with homogeneous class distribution and lower degree of impurity are preferred). Gini index over K classes is defined as:

$$Gini = \sum_{i=1}^{K} p_{mk}(1 - p_{mk})$$

where p_{mk} represents the proportion of training observations in the mth region that are from the kth class. A small value indicates that a node contains predominantly observations from a single class.

When building a decision tree, one should consider a **trade-off** between fitting data well with a large and deep tree and avoiding overfitting. In fact, the larger and deeper a decision tree is, the better its accuracy on training data, but the lower its generalization capability. In other words, since each leaf represents a class, a tree with k leaves can shatter any set of k instances. If we do not limit in some way the number of leaves, letting the tree growing indefinitely, the VC dimension becomes infinite and the method would be prone to overfitting, out of PAC learning framework. Thus, some measures have to be taken such as setting maximum depth of the tree or pruning the tree.

4.1.1 Bagging trees and Random Forest

In order to reduce the variance introduced by the complexity of a tree (e.g. its depth), the *bagging* technique can help. In addiction to bootstrap, that can enforce the different trees to be almost uncorrelated, a *feature bagging* approach can be introduced to enhance this constraint. More specifically, at **each** split in each tree, a random subset (of size s) of the d features is selected as candidates for the split, rather than selecting among all the features.

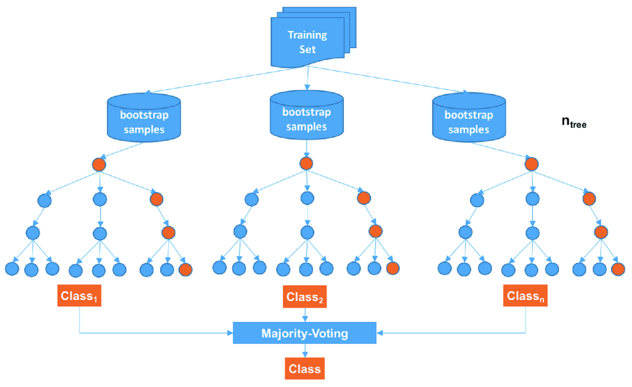

Figure 14: A random forest made up of three trees, following a majority voting policy.

In fact, it may happen that one or few predictors are so strong that they will be selected by most of the trees, without exploiting all the other predictors and making some trees more correlated than others. This ensemble of trees with this kind double bagging technique applied to decision trees is named **Random Forest**, and usually a good number of features is $s = \sqrt{d}$.

4.1.2 Boosting Trees

As previously stated, the *boosting* approach mainly aims at reducing the bias, not the variance. Moreover, it requires simpler (and thus possibly weaker) learners since it cannot work in parallel.

For all these reasons, boosting very deep trees could not bring to good performances. Contrariwise, we can use simpler trees (with a lower depth) characterized by an high bias and a low variance, that are more suited to be boosted.

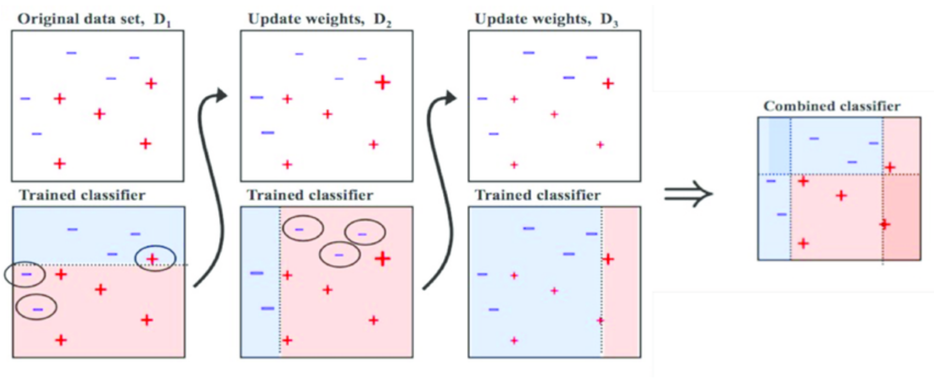

Figure 15: The AdaBoost algorithm applied to stumps (trees with only 1 split). Each stump divides the feature space into two parts, defining a sort of dividing hyperplane parallel to one of the axis.

4.2 Support Vector Machines

SVM is a supervised learning algorithm that tries to find a separating hyperplane between two (or more) classes. Hard-margin SVM seeks the halfspace that separates the data perfectly with the largest margin, that is minimal distance between a point in the training set and the hyperplane. Soft-margin SVM, instead, does not assume separability of data and allow the constraints to be violated to some extent.

4.2.1 Hard margin vs soft margin

In a binary classification case, where labels $y_i \in \{\pm 1\}$, when training data $(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_m}, y_m)$ are linearly separable, in other words there exist a half-space defined by $(\mathbf{w}, b)$ such that $\forall i \in |m|$, $y_i(\mathbf{w}\mathbf{x_i} + b) \geq 1$, we can select two parallel hyperplanes that separate the two classes of data, so that the distance between them is as large as possible (all points can be classified correctly).

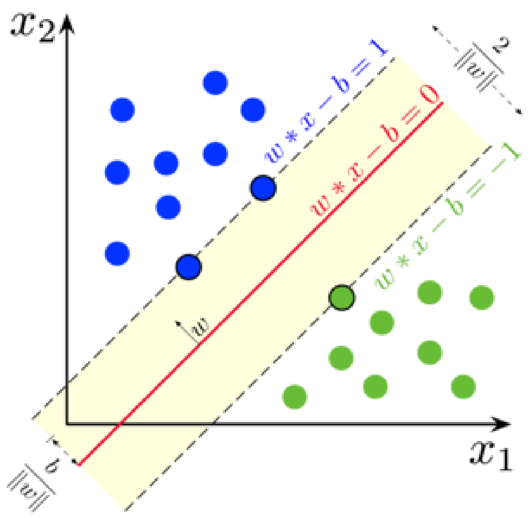

Figure 16: Maximum-margin hyperplane and margins for an SVM trained with samples from two classes.

The region bounded by these two hyperplanes is called the *margin*, and the maximum-margin hyperplane is the hyperplane that lies halfway between them. Geometrically, the distance between these two hyperplanes is $\frac{2}{||w||}$, so to maximize the distance between the planes we want to minimize $||w||$. The corresponding optimization problem is:

$$\underset{\mathbf{w},b}{\text{minimize}} \ \frac{1}{2}||\mathbf{w}||^2$$
$$s.t. \quad y_i(\mathbf{w}\mathbf{x_i} + b) \geq 1$$

and the max-margin hyperplane is completely determined by those vectors that lie nearest to it. These x_i are called *support vectors*.

Most of the times data are not linearly separable because of noise, thus we introduce non-negative slack variables $\xi_1, ..., \xi_m$ and rewrite the problem as follows:

$$\underset{\mathbf{w},b}{\text{minimize}} \ \frac{1}{2}||\mathbf{w}||^2 \ + C\sum_{i=1}^{m}\xi_i$$
$$s.t. \quad y_i(\mathbf{w}\mathbf{x_i} + b) \geq 1 - \xi_i$$
$$\xi_i \geq 0$$

ξ_i measure how much the constraint is being violated. Soft-SVM objective function jointly minimizes the norm of $\mathbf{w}$ and the violations of the constraints that we are allowing. C is a regularization parameter of the classification algorithm which is proportional to the error that we allow because it multiplies the slack variables. Thus, C tells me how much the margin is soft (larger C) or hard (lower C).

4.2.2 Kernel trick

When softening the margin is not enough for making a proper classification, we would like to map training data into another feature space, usually of higher dimensions, thanks to a certain mapping Φ in order to make training data separable. This procedure ends up being theoretically appropriate but practically too computational demanding. Since the solution of the SVM optimization problem can be written as a function of inner products, we can use kernel functions,

$$K(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$$

in order to make the algorithm non-linear. The only thing we should know becomes how to calculate inner products between the mapped instances in the higher feature space, or equivalently to calculate the kernel function without explicitly applying the transformation Φ and without using instances' representation in that space. There exist several types of kernel functions, the linear or the Gaussian (Radial Basis Function) are the most used.

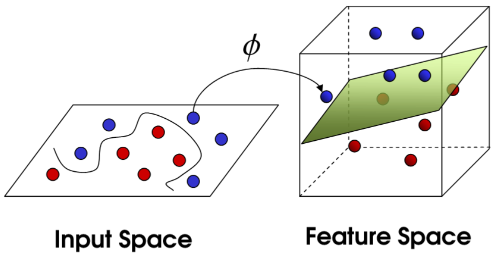

Figure 17: Mapping Φ from input space to feature space in order to find separating SVM hyperplane

5 Results

In this results section, I show the practical implications of the bias-variance trade off on decision trees (particularly with respect to *max_depth* parameter) and how ensemble techniques can handle them. Then, a similar analisys is done on SVM for which the regularization parameter C and the kernel type are tuned. Accuracy will be used as metric and confusion matrices will be shown in order to inspect models' behaviour with respect to the unbalancing of the dataset.

The code is available on my GitHub at: `https://github.com/giocio/MML_project`

5.1 Decision Trees

First of all, I trained single models with DecisionTreeClassifier (from *scikit-learn* library) with Gini impurity parameter and increasing values of *max_depth* and I tested it on both training set and test set in order to inspect the presence of overfitting. The plot in Figure 18 clearly shows the different behaviours of the accuracy curves: as the maximum depth increase, the performances of the model on the training set are better, since it is building more complex and deep trees and learning how to perfectly classify all points (in fact it arrives at accuracy of 1), but the performances on the test set get worsening. This demonstrates that a single weak tree is prone to poorly generalize as the tree increase in size. Moreover, a single decision tree does not overcome 92% accuracy score on the test set.

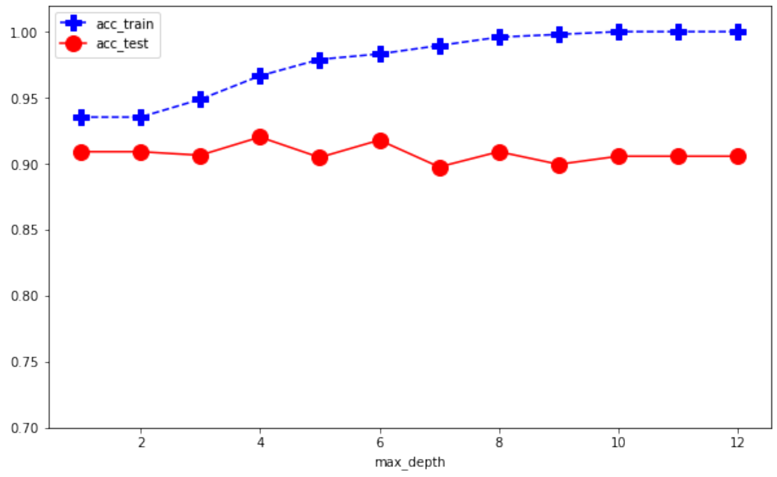

Figure 18: Train accuracy and test accuracy of a single decision trees.

Thus, I analysed the performances of ensembles of trees. I used RandomForestClassifier, BaggingClassifier and AdaBoostClassifier from *scikit-learn* with DecisionTreeClassifier as base estimator with Gini Index as impurity measure. I varied again the maximum depth with a fixed number of estimators $n\_estimators = 100$.

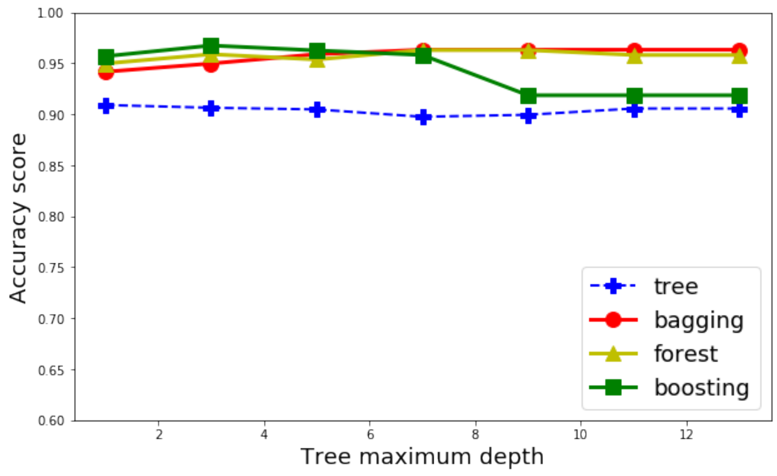

Figure 19: Ensemble trees performances varying maximum depth with $n\_estimators = 100$

From the accuracy graphs we can extract some insights:

- each ensemble technique performs significantly better than a simple decision tree (blue crossed line);
- boosting with AdaBoostClassifier works better with threes with a lower maximum depth, expecially with maximum depths from 1 to 5. This confirmed its ability to reduce the bias and correct the mistakes (green squared line);
- contrary to boosting, BaggingClassifier (red circled line) and RandomForestClassifier (yellow triangled line) work better when the tree is deeper, that is when the model has more variability and prone to overfit;

Finally, the best performing ensemble model was AdaBoost with 96% accuracy score and on the test set. The confusion matrix shown below demonstrates that the unbalancing between the two classes has a slight effect on performance. In fact, there 4 records classified as benign that should have been classifier as malign.

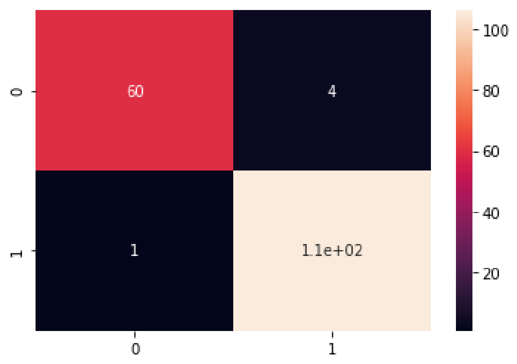

Figure 20: Confusion matrix of AdaBoost of trees

5.2 SVMs

SVM is commonly known as a strong learner. Thus, it should not significantly benefit from ensemble models.

First, I used a K-fold cross-validation in order to choose the best C (penalty) and type of kernel parameters between *linear* and *rbf*. The single base model with the best configuration found (linear kernel and $C = 0.1$) was trained with those parameters and used as base estimator for bagging and boosting.

Figure 21 shows that the performances of BaggingClassifier and AdaBoostClassifier with SVC as base estimator stay stable with respect to n_estimators, but none of them succeed in significantly improving accuracy. Bagging seems to slightly improve accuracy whereas boosting has worst performances with respect to single SVM.

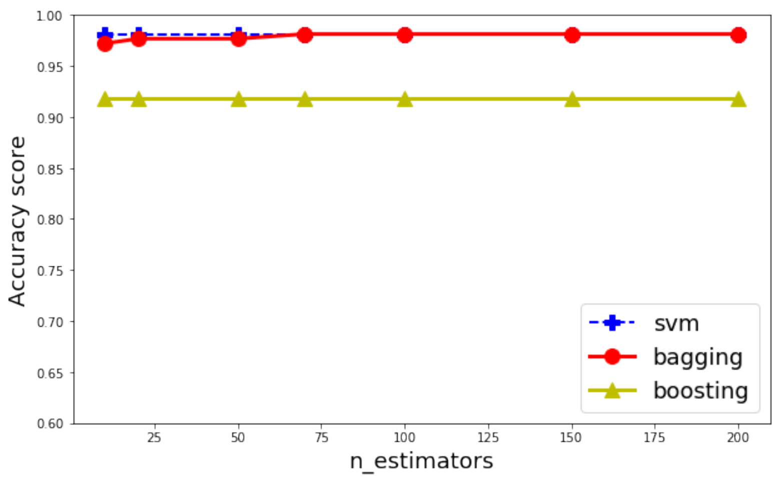

Figure 21: Boosting and Bagging applied to SVM performances on test set

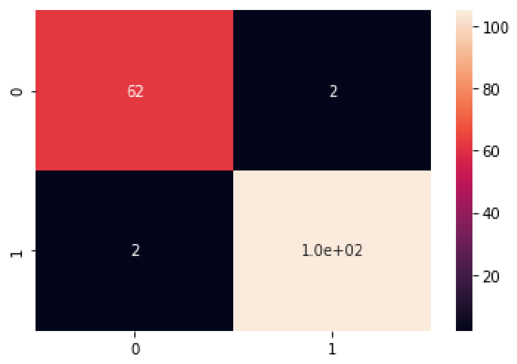

Figure 22: Confusion matrix of SVM Bagging

Support Vector Machine bagging seemed to work better that tree boosting in tackling class unbalancing. In fact, false positive are equal to false negatives and they are only 2. The final accuracy score on the test set for SVM bagging was 98%.

The final consideration is that ensemble techniques can be very useful to tackle the bias-variance trade-off introduced when learning through tree-based weak models and can improve their performances. SVM as strong learner, instead, does not significantly benefit from those techniques.