Project 3

Eirik Ramsli Hauge
Joakim Kalsnes

December 13, 2018

1 Introduction

Throughout this course we have studied numerous different machine learning methods for both regression and classification analysis. In this project we will use the classifications methods we have learned on real-life data. We will study the credit card data set presented at the site of UCI [reference UCI, link i oppgavetekts] . The data set consists of 25000 credit card observations taken in 2005 from an important bank in Taiwan. In order to get more costumers, banks in Taiwan lowered the requirements for credit card approval. Cardholders, at the same time, overused their credit cards and accumulated heavy debts.
This study will try to use information such as business financial statement, repayment records and personal information (gender, age, education, marital status) to predict whether or not the costumer will be able to repay their credit card debt. To make our predictions, we will use logistic regression, neural networks and random forest. The calculations will be performed using the Scikit-Learn and TensorFlow packages. The credit card data set has previously been studied by Yeh and Lien [reference article] , so it will be natural to compare our results to theirs. However, random forests was not tested by Yeh and Lien, so it will be interesting to see how these results compare to the other classification methods.

2 Theory

For this project we are going to use logistic regression, neural networks and random forest to analyse the credit card example. Logistic regression and neural networks were described in project 2 (reference). This theory section will therefore focus on random forest, starting with a presentation of decision trees.

2.1 Decision Trees

This section will follow closely the description of decision trees in chapter 6 of Géron's book [reference].
Decision trees is a type of Machine Learning algorithm that can be used for both regression and classification. They are easy to interpret and intuitive in the way the algorithm is constructed. As we will see in the next section, they are the fundamental components of random forests. To understand how decision trees work, we start by looking at an example. Figure 2.1 shows a decision tree made on the classic iris dataset using Scikit-Learn (taken from Géron p. 170 [reference]). The goal of the model is to classify an iris flower. When we have a new flower, we start at the *root node* (depth=0) and ask if the petal length is smaller than 2.45 cm. If it is, we move down the tree to the left child node (depth=1, left). In this example, it is a leaf node, that is it does not have any children nodes. From this lead node we see that the decision tree predicts that the flower is and Iris-setosa (class=setosa). If the petal length is longer than 2.45 cm we move to the root's right child node (depth=1, right). This is not a leaf node, so it asks a new question: is the flower's petal width smaller than 1.75 cm? If it is, then the flower is most likely of class Versicolor. If it is not, it is most likely of class Virginica.

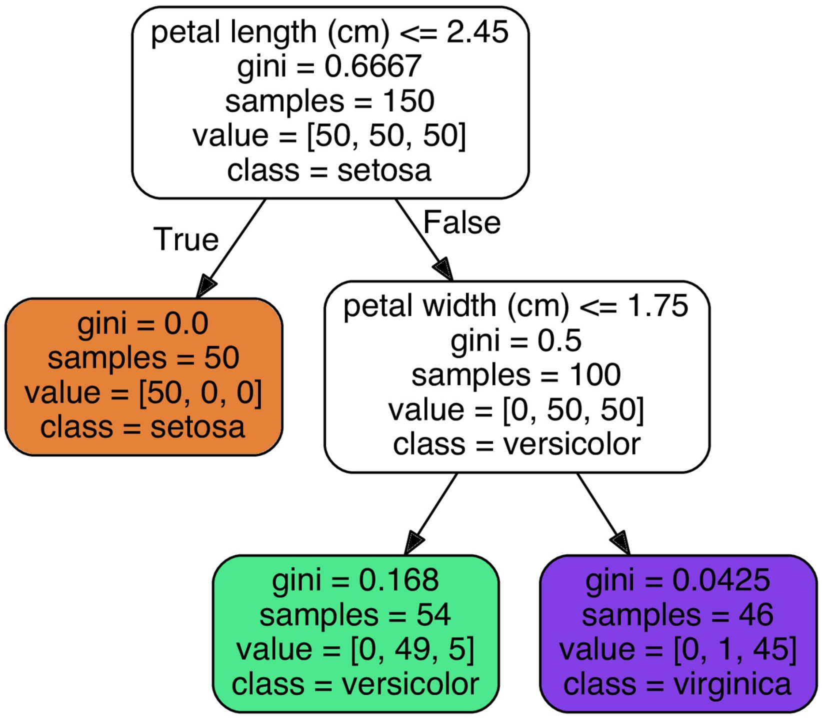

Figure 2.1: Iris Decision Tree

A node's *samples* attribute counts how many training instances it applies to. For example, in Figure 2.1, 50 training instances have a petal length smaller than 2.45 cm. The *value* attribute tells us how many training instance of each class this node applies to. For example, the bottom left applies to 0 Iris-Setosa, 49 Iris-Versicolor and 5 Iris-Virginica. The *class* gives the most likely class, i.e. the class with the highest count in *value*. We can also get the probability of each class as an output (not shown in Figure 2.1). The probability is simple the ratio of class instances among the training instances in the given node. For example in (depth 2, left), the probabilities are: p(class=setosa) = 0/54, p(class=versicolor) = 49/54 and p(class=virginica) = 5/54. Finally, a node's *gini* attribute measures its *impurity*. A node is "pure" (*gini* = 0) if all training instances it applies to belong to the same class, e.g. (depth 1,left) in figure 2.1. The gini score, G_i, is calculated using the following equation:

$$G_i = 1 - \sum_{k=1}^{n} {p_{i,k}}^2, \tag{1}$$

where $p_{i,k}$ is the ratio of class k instances among the training instances in the i^{th} node. As an example, the gini score of (depth 2, right) is: $1-(0/46)^2-(1/46)^2-(45/46)^1 \approx 0.0425$.

To train, or "grow", decision trees, Scikit-Learn uses the *Classification and Regression Tree* (CART) algorithm. The algorithm first splits the training set in two subsets using a single feature k and a thresholdt_k. To choose k and t_k, the algorithm searches for the pair (k, t_k) that produces the purest subsets. The cost function to minimize is given by the following equation:

$$J(k, t_k) = \frac{m_{left}}{m} G_{left} \; + \; \frac{m_{right}}{m} G_{right}, \tag{2}$$

where $G_{left/right}$ measures the impurity of the left/right subset and $m_{left/right}$ is the number of instances in the left/right subset. The Gini impurity measure (equation 1) is used by default, but one can also use entropy impurity defined by:

$$H_i = -\underbrace{\sum_{k=1}^{n} p_{i,k} log(p_i i, k)}_{p_{i,k} \neq 0}, \tag{3}$$

where $p_{i,k}$ is the same as in equation 1. Once the algorithm has split the training set in two subsets, it splits the subsets using the same logic, then the sub-subsets and so on. The algorithm will stop when it cannot find a split that reduces the impurity or when it reaches the maximum depth, which is a hyperparameter set by the user.

Decision trees make very few assumptions about the training data. If no constrain is set on the algorithm, it will likely adapt the tree structure very well to the training data. This will however most likely lead to overfitting, and the tree will generalize poorly to test data. To avoid overfitting, we therefore need to restrict, regularize, the decision tree. One way is by restricting the maximum depth of the tree, where a lower maximum depth reduces the risk of overfitting. Other parameters that restrict the shape of the decision tree is:

• Minimum number of samples a node must have before it can be split.
• Minimum number of samples a leaf node must have (absolute number or fraction of total instances).
• Maximum number of leaf nodes.

• Maximum number of feature that are evaluated for splitting at each node.

We will only use decision tress and random forests for classification, but we mention briefly here that can be used for regression as well. The algorithm, and hence the decision trees, are very similar to the classification case. The regression prediction will simply be the average value of the training instances in the leaf node you end up in. The splitting is now done in way that minimizes the mean squared error (MSE) rather than the impurity.

The main issue with decision trees is that are very sensitive to small variations in the training data. Just removing or adding one sample, may change the tree completely. In addition, since the Scikit-Learn model is stochastic we might get very different models on the same training set. Random forests limits this instability by averaging over many trees.

2.2 Random Forests

This section is heavily based on chapter 7 about ensemble learning and random forests in Géron's book [reference].
The idea behind random forests is simple: You train a group of decision trees, each on a different random subset of the training set. To make predictions, you obtain the predictions from all individual trees, then predict the class that gets the most votes. Random forests is an *Ensemble method*, where an *ensemble* in machine learning is a group of predictors.

When making the prediction, there are different ways of counting the votes. One can use either *hard voting* or *soft voting*. Hard voting is the most straightforward method where you just count the number of predictors that predict each class, and choose the class that got the most votes. In soft voting, we use the probability of each class. The class probabilities are found from averaging the predicted probability over all individual predictors. To predict the class, we then use the class with the highest class probability. Soft voting gives more weight to highly confident votes, and therefore often achieves higher performance than hard voting.

Like discussed above, random forest is an ensemble of decision trees. The

trees are usually trained using the bagging method, or sometimes pasting. Both of these methods use the same training algorithm for every predictor, but trains them on different random subsets of the training set. Bagging uses sampling with replacement (bootstrap), while pasting samples without replacement. The bias is higher for each individual predictor than if it were trained on the complete training set, but aggregation reduces both bias and variance. The result is generally a similar bias but a lower variance for the ensemble compared to a predictor trained on the original training set. This means that the ensemble's (random forest) prediction will generalize better than the single decision tree's prediction.
In addition to sampling training instances, the random forest algorithm samples features. Instead of searching for the very best feature when splitting a node, it searches for the best feature among a random subset of features. This also results in higher bias and lower variance, making a more generalizable and overall better model.

In our project we will use the *RandomForestClassifier* class from Scitkit-Learn. When using *RandomForestClassifier*, we can adjust all the hyperparameters described in the Decision Tree section, as well as hyperparameters controlling the sampling.
A great quality of random forest is that they make it easy to measure the relative importance of each feature. "Scikit-Learn measures importance by looking at how much the tree nodes that use the feature reduce impurity on average (across all trees in the forest). More precisely, it is a weighted average, where each node's weight is equal to the number of training samples that are associated with it." [Géron, p.192]

3 Discussion

Interpretability of models. White box vesus black box (Géron p. 172).