Introduction

The main focus of the project was to get a basic intuition and first-hand experience of two machine learning algorithms for classification. Classification is a very popular task both in business and academia, thus it is crucial for every data science specialist to know how to do it right. In this project, we used two algorithms: a decision tree and support vector machine. Both algorithms are very popular and are used daily by every specialist in the field. To see how they work, we used Titanic and Wisconsin Breast Cancer data sets. In terms of programming languages, we chose Python as it is the most popular language in machine learning at the moment. Just like in a real-life task we did not implement our algorithms from scratch, instead, we used a very popular and well-documented library called sklearn.

Description

- *Task 1 - The decision tree algorithm for classification on the Titanic data set*
  I. Theory

The algorithm of this part is a decision tree. The decision tree as a machine learning algorithm is essentially incorporating a stream of logical rules of the form "feature a value is less than x and feature b value is less than y ... => Category 1" into a tree-like data structure. For our algorithm to see which column to choose and where it should be split to form such a rule we can use the concept of information gain based on entropy. Shannon's entropy is defined for a system with N possible states as follows:

$S = -\sum_{i=1}^{N} p_i \log_2 p_i$,where p-i is the probability of finding the system in the i-th state.

Entropy can be described as the degree of chaos in the system. The higher the entropy, the less ordered the system and vice versa. This will help us formalize "effective data splitting". The advantage of this algorithm is that they are easily interpretable. For example, the bank can explain to the client why they were denied for a loan: e.g the client does not own a house and her income is less than 5,000.

The most important thing in every machine learning model is how well does this model generalizes (performance on unseen data). Since we cannot immediately check the model performance on new, incoming data (because we do not know the true values of the target variable yet), it is necessary to sacrifice a small portion of the data to check the quality of the model on it.

This is often done in one of two ways:

- setting aside a part of the data set (held-out/hold-out set). Thus we reserve a fraction of the training set (typically from 20% to 40%), train the model on the remaining data (60-80% of the original set), and compute performance metrics for the model (e.g accuracy) on the hold-out set.
- cross-validation. The most frequent case here is k-fold cross-validation.

In k-fold cross-validation, the model is trained K times on different ($K-1$) subsets of the original data-set (in white) and checked on the remaining subset (each time a different one, shown above in orange). We obtain K model quality assessments that are usually averaged to give an overall average quality of classification/regression.

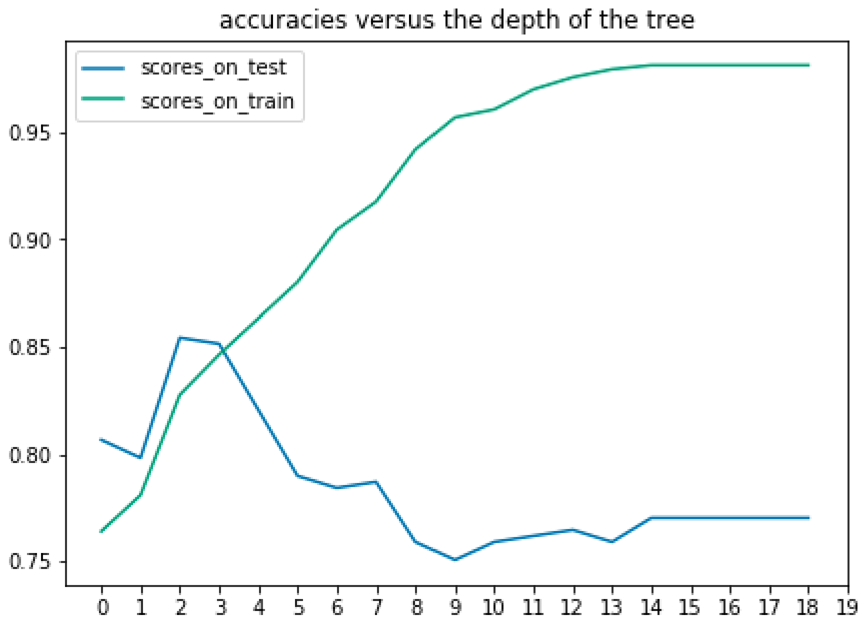
Cross-validation provides a better assessment of the model quality on new data compared to the hold-out set approach. However, cross-validation is computationally expensive when you have a lot of data. It is a very important technique in machine learning and can also be applied in statistics and econometrics. It helps with hyper-parameter tuning, model comparison, feature evaluation, etc. In our task, we have used both hold-out (40%) and cross-validation technique.

Before training our first model, we need to understand one more theoretical concept – overfitting. Technically, you can build a decision tree until each leaf has exactly one instance, but this is not common in practice when building a single tree because it will be overfitted, or too tuned to the training set, and will not predict labels for new data well. At the bottom of the tree, at some great depth, there will be partitions on less important features (e.g. whether a passenger has blue eyes). Even if that were true in training, we do not want our classification model to generate such specific rules. The most common way to deal with overfitting in decision trees is an artificial limitation of the tree's depth.

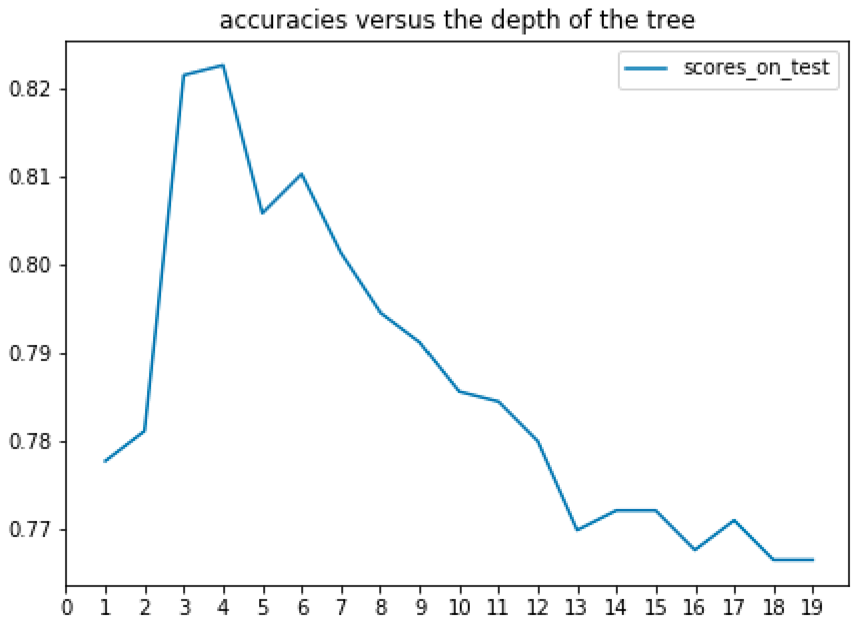
II. Practice

Doing the first task, we have come across a couple of challenges. The first of them were missing values in the data set. To fix it we filled the missing parts with the median value of the column if the column was continuous and with the mode value if it was categorical. To improve the data set we also added some obvious features. The second challenge was categorical columns. It is very uncommon (and in most cases incorrect) to fit raw categorical columns into machine learning algorithms, thus we have encoded every such column (new integer for every category).

Firstly, we trained our decision tree classifier with default parameters and gained 77% accuracy on the test set and 98% accuracy on the train set. It is a classical example of overfitting. To combat that we have decided to tune the tree's depth. We got such results:



Looking on this chart we can think that the best tree depth for this task would be a depth of 2 but as we already know, we cannot put a lot of trust in a simple hold-out set experiment. Thus, we do the same but using 5 fold cross-validation:



Now, we can see that actually, the best depth of the tree is 4. After we trained our model with depth 4, we got 82% cross-validated accuracy, which is a huge improvement compared to the previous 77%.

- *Task 2 - Support Vector Machine to perform classification a task on the Wisconsin Breast Cancer data set.*

  I. Theory

The algorithm of this part is the Support Vector Machine. Support-vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin, the lower the generalization error of the classifier. Whereas the original problem may be stated in a finite-dimensional space, it often happens that the sets to discriminate are not linearly separable in that space. For this reason, it was proposed that the original finite-dimensional space be mapped into a much higher-dimensional space, presumably making the separation easier in that space. To keep the computational load reasonable, the mappings used by SVM schemes are designed to ensure that dot products of pairs of input data vectors may be computed easily in terms of the variables in the original space, by defining them in terms of a kernel function.
If the training data is linearly separable, we can select two parallel hyperplanes that separate the two classes of data, so that the distance between them is as large as possible. The region bounded by these two hyperplanes is called the "margin", and the maximum-margin hyperplane is the hyperplane that lies halfway between them.

To visualize the relations between the features we used scatter plots. A scatter plot is a type of plot or mathematical diagram using Cartesian coordinates to display values for typically two variables for a set of data. If the points are coded (color/shape/size), one additional variable can be displayed. The data are displayed as a collection of points, each having the value of one variable determining the position on the horizontal axis and the value of the other variable determining the position on the vertical axis.

To see how well our SVM model worked, we used a concept called a decision region of the classifier. A decision region: is a specific region within the input space that corresponds to a unique output class. All points within this region contain one and only one output class. Note that the input space can have multiple decision regions corresponding to multiple output classes.

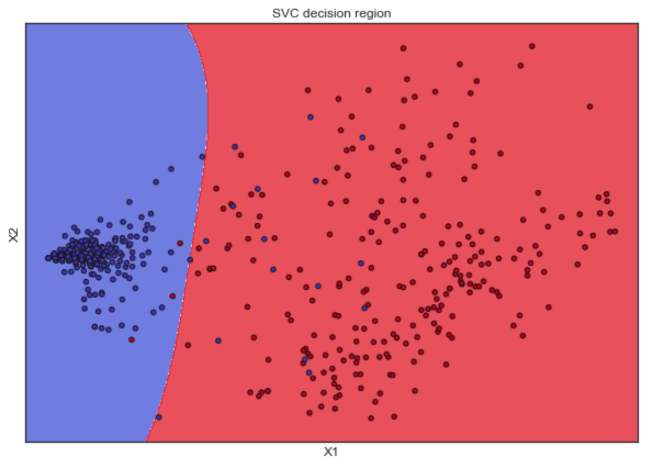
To plot our decision region, we needed to have a 2-d input space (only 2 features) not 11-d like in our data set. To reduce dimensionality we have used a technique named PCA. Given a collection of points in two, three, or higher dimensional space, a "best fitting" line can be defined as one that minimizes the average squared distance from a point to the line. The next best-fitting line can be similarly chosen from directions perpendicular to the first. Repeating this process yields an orthogonal basis in which different individual dimensions of the data are uncorrelated. These basis vectors are called principal components, and several related procedures principal component analysis (PCA).

II. Practice

Like in the first data set in this one we also have some missing values in 'Bare Nuclei' column. We filled the missing parts with the mode value of the column. After that, we plotted relations between every column (feature) using scatter plots.

In terms of model performance, we have conducted both hold-out and cross-validation experiment. Using only default parameters, we have achieved an incredible result – 97% accuracy on hold-out (test) set, which is a truly awesome performance. To double-check the performance, we tried cross-validation and got an accuracy of 96%, which is still an extraordinary result.

To see the decision region of the classifier, we transformed our data set in two-dimensional feature space using principal component analysis (PCA) technique. It is worth mentioning that such a transformation is impossible without an information lost. In this particular transformation we lost 36% of information, which is not so bad recalling that we have gone from 11-d to 2-d feature space. Finally, we got the beautiful plot of just how well our SVM separated the classes:



Implementation

In terms of implementation of all the algorithms and techniques described above, we used just 5 libraries: NumPy, pandas, matplotlib, seaborn and sklearn. These libraries are essential to know as they are used in real-life tasks daily.

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. All the mathematical operations in the project (taking a mode, median, mean of the column values), were done by using NumPy.

Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. All the data manipulations in the project (feature engineering, filling missing data, displaying the tabular data, etc.), were done by using pandas.

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI tool-kits like Tkinter, wxPython, Qt, or GTK+. There is also a procedural pylab interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB. All the data manipulations in the project, were done by using seaborn and matplotlib.

Scikit-learn (also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. We have used such skleran’s classes in the project:

- DecisionTreeClassifier() – for a decision tree classifier implementation
- PCA() – for a principal component analysis implementation
- SVC() – for a Support Vector Machine classifier implementation
- train_test_split() – to make a hold-out data-set
- cross_val_score() – to implement cross-validation
- accuracy_score() – to check the accuracy of the model

Results and Conclusions

In this project, we solved classification problem on two real-life data-sets. We had to done some preprocessing and feature engineering to both of them before fitting into the models.

On the first data-set we used a decision tree classifier, with the default parameters it struggled with overfitting (you can clearly see it on the plot above in the report) so we have conducted a fine-tuning of the tree depth parameter. After tuning the parameter, we achieved an accuracy of 82%, which is a good result for such a task. Although, one can gradually improve the classification accuracy on this data-set by doing more in-depth feature engineering and using more powerful models, such as: random forest or gradient boosting.

On the second data-set we used a Support Vector Machine classifier. We have done some visualization to check if columns do not correlate to much (it is very bad for linear models). After training the model with the default parameters we achieved an accuracy of 97% on a hold-out set and a 96% on a cross-validation, which is an outstanding result for such a task.

To sum up, I can say that it was a very interesting project that gave me a vital intuition and experience with such an important task like classification.

References and Citations

- https://mlcourse.ai/ - algorithms, concepts and techniques explanations
- https://numpy.org/ - NumPy documentation
- https://pandas.pydata.org/ - pandas documentation
- https://scikit-learn.org/stable/- sklearn documentation
- https://seaborn.pydata.org/- seaborn documentation
- https://matplotlib.org/- matplotlib documentation