Data Stewardship - Exercise 1

Warren Purcell

1 Introduction

The purpose of this report is to compare three different classifiers through supervised machine learning on two diverse datasets. The whole machine learning process was applied and conducted different experiments. The exploration of the data sets as well as the preprocessing strategies are outlined in the following. Furthermore, the modelling processes and the performance measures on which their results are evaluated will be explained. Finally, different parameter adjustments and settings are compared and discussed which leads to a conclusion.
Supervised learning trains the model on known input and output data to predict future outputs. The training will be done using supervised classification (pattern recognition) to predict the nominal class labels. One of the smaller datasets is a set of 690 datapoints about the binary classification of credit-approval with 16 dimensions and missing values. Another small set contains 218 instances of the binary class labels of American congress party members described by 16 binary features. The Location dataset is much larger with 446 binary features describing 4000 instances and 30 types of the target attribute. The second large dataset about the quality of red wine consists of 1600 samples with 12 dimensions and without missing values.
These specific datasets were chosen in order to have diversity across number of samples, dimensions and classes. The credit-approval dataset and the congressional voting dataset are smaller datasets. One might notice the similarity regarding these characteristics for the credit-approval and the congressional voting dataset. However, the credit-approval dataset was chosen, because the approach for this data might still be different from the others due to the underlying anonymization. Preprocessing is needed across all datasets but especially for this one.

2 Datasets

The following gives an overview of the utilized data in order to better understand applied approaches.

2.1 Congressional Voting

`https://www.kaggle.com/c/184702-tu-ml-ws-21-congressional-voting/data`

2.1.1 Description

The training set includes 218 listings of the binary classes represented by the political democratic and republican party. Each data-point consists of an ID, the class and their binary values of 16 features. The 16 features are political matters to which each ID has a specific attitude expressed by yes (y) or no (n). These categorical features of nominal value type can be considered as labels

Table 1: Dataset overview

Number of classes	2
Number of features	16
Number of training set instances	218
Number of test set instances	217
Number of instances of class "democrat" in the training set	125
Number of instances of class "republican" in the training set	93
Number of instances in the training set that hold missing values	107
Number of missing values across the training set	199

and cannot be ordered or measured.
The split into training and test set is almost even, with a difference of one instance more in the training set. With around 57% of the training data being covered by the class "democrat" and the other 43% by the class "republican", the classes can be considered as quite balanced. The total number of 199 missing values within the training set affects 107 instances. Most of the data can be considered as accurate, but one data-point does not hold any information since all its features are described by unknown values.

2.1.2 Goal

The task of this model is to predict the label "democrat" or "republican" for a given ID and their answers to political questions in the format of yes or no. The output of the classification algorithm is the ID and political party.

2.1.3 Pre-processing

The data was already consistent in its structure and format and split into training and test data when downloaded from kaggle before any pre-processing steps. It is very hard to detect outliers in this dataset since there are not many rows and the features are binary.
First of all, all binary values y and n were exchanged with numeric values 1 and 0, because the algorithm needs numeric values to calculate with.
Instances with a lot of missing values are not very helpful, since they do not contribute much information on the class. This is why all rows with more than half of the attribute values missing were removed from the dataset. It affected two data points.
Some matters that are noticed during data exploration are not very clear how to be preprocessed at this stage and need to be tested with the help of the classifiers first. For example the column 'export-administration-act-south-africa' has the most missing values for both training set (50) and test set (54). That is roughly one quarter of the data set. Only about one quarter (11) of these data points belong to the class 'republican'. The incompleteness of this feature may imply that it is not relevant to the defined task. There is also the possibility that the model takes a missing value for the attribute 'export-administration-act-south-africa' falsely into account as the data point being linked to the class 'democrat'. It should be tested whether the feature is misleading and should be

removed, or whether a removal is too great of an information loss. The columns with the second highest missing values 'water-project-cost-sharing' (20) and below are rather equally split between the classes and are therefore not considered to have a negative influence on the model.
Finally, the missing values need to be filled in. Two possibilities are considered. They can either be replaced randomly with 0 or 1 or consistently with -1.
An important preprocessing step is also to exclude leaky variables from the training and test set. The ID could be an index rather than a relevant information on the class which would make it a leaky variable that should be removed.
Furthermore, the Scikit-Learn Standardscaler was used to scale the values and the binary classes were encoded before training.

2.2 Credit-approval

`https://www.openml.org/d/29`

Table 2: Data set overview

Number of Instances	690
Number of Features	16
Number of Classes	2
Number of Missing Values	67
Number of Instances with Missing Values	37

2.2.1 Description

The dataset is comprised of 690 instances of anonymized (I.e., transformed feature names and values) credit card applications, with sixteen columns consisting of fifteen features and one class label (either positive or negative).
These features can be broken down respectively as six (A2, A3, A8, A11, A14, A15) numerical continuous values, five (A4, A5, A6, A7, 13) large-nominal (>2) and five (A1, A9, A10, A12, class) binary-nominal values including the class label. Thirty-Seven rows or about 5% of instances are missing one or more feature values, with A1, A2, A14 being the worst effected. The distribution of the labeled classes is slightly skewed, with 307 (44.5%) positive and 383 (55.5%) negative instances respectively.
This dataset was chosen due to the unique anonymisation of the input variables, as well as due to the amount of missing values.

2.2.2 Goal

The task of this model is to predict whether a credit application is approved or not. The performance will be evaluated utilizing methods such as Classification Accuracy, Confusion Matrix, F1 Score, etc.

2.2.3 Preprocessing

Since the data set only includes 690 values, dropping 5% of rows containing missing values would significantly reduce the data available for training. This necessitates the imputation of values which was done using the mean for numerical input features and most frequent for categorical features. The mixture of float and alphanumeric characters necessitates the use of OneHotEncoder. This causes a substantial increase in dimensions from 16 to 54. Again a z-score scaler was deployed on the input features.

3 Classifiers

Three different classifiers were chosen and are introduced in the following.

3.1 Random Forest

The random forest algorithm expands on the idea of the decision tree. By training multiple trees on a subset of the input features, the overfitting associated with decision trees is reduced. Random forest uses bagging in the training process. The final classification is done via majority voting of the trees. Since feature selection is an intrinsic part of the algorithm, random forest should perform well on high dimensional data. Random forest was chosen because it is a well established algorithm that is known to produce reliable results especially for high dimensional data sets. This makes random forest an suitable approach for the location data set.

Hyper parameter tuning was done on three different parameters:

- n_estimators: controls the amount of trees in the forest. A higher value increases predictive power but might lead to overfitting.
- max_depth: limits the depth of each tree. Suitable to reduce overfitting.
- max_features: size of the feature subset.

3.2 Support Vector Machine

The Support Vector Machine, abbreviated as SVM can be used for both regression and classification [3]. It originated from Statistical Learning Theory (SLT) where the objective is to find a loss function that minimizes the expected prediction error [2].
The approach of the pattern classification algorithm is to calculate a hyperplane as a decision boundary that separates the data into distinct classes. First the input data is mapped into a N-dimensional feature space with N meaning the number of attributes. Respectively the dimension of the hyperplane depends on the number of attributes. Then the algorithm tries to find a plane with the maximum margin so that the distance between data points of the different classes gets maximized. This gives the model more confidence when generalizing on unseen data. The data points

that define the position and orientation of the hyperplane are called support vectors and are closest to it [3]. For classification problems of non-linear separable data the SVM uses Kernel functions to transform non-linear spaces into linear spaces and turn non-separable data into separable data [1]. The classification is based on the output of the linear function of which the threshold values are 1 and -1 which conveys a large margin intuition [3].

This classifier was chosen, because it is a simple algorithm that should be able to effectively separate our non-overlapping classes, especially in high-dimensional feature spaces [6]. Furthermore, the advantages of SVM are the simplicity of the algorithm, using less computational power while producing significantly accurate results [3].

The algorithm is usually applied on binary classification problems of which we are considering two in this report, but can also be applied to multiclass tasks using a one-versus-all approach [1]. The two major parameters to be tuned for this classifier are C and Gamma. The higher the C value the more data points are represented correctly by the decision boundary. However, there is a trade-off between a smooth decision boundary and the correct classification of all data points and a risk for over fitting. Gamma on the other hand weights the data points that are close to the decision boundary which determines the shape of the boundary. A high value for gamma ignores some of the data points further away and gives the data points close to the decision boundary more influence. A low value will make the boundary less dependent on the close points and also takes the points further away into account [7].

3.3 Multi-layer Perceptron

The Multi-Layer Perceptron, abbreviated as MLP is a feed-forward neural network with back-propagation. Consisting of an input layer, output later and zero or more hidden layers. The MLP can be used for either classification or regression tasks, with its ability to model non-liner problems makes it extremely useful [5]. This model optimizes the log-loss function using LBFGS or stochastic gradient descent, by calculating the partial derivative at each training iteration back-propagation can be used to update the weights. However, optimizing the layer width and depth of the MLP network still remains a time consuming task [8]. The calculated weights in combination with an activation function (e.g., rectified linear units (ReLU), sigmoid function, tanh, etc) provide a threshold mechanism for neuron activation.

Hyperparameter optimization was investigated utilizing:

- GridSearchCV: This is an exhaustive search methods to explore specified parameter values for an estimator, it uses “fit” and a “score” approach.
- RandomizedSearchCV: In contrast to GridSearchCV, not all parameter values are tried out, but rather a fixed number of parameter settings is sampled from the specified distributions.

The MLP hyperparameters that were investigated utilizing the above hyperparameter optimization approaches are:

- max_iter: The maximum number of allowed iterations

- hidden_layer_sizes: This is a tuple which sets the hidden layer width and depth
- solver: The solver is algorithm for weight optimization

GridSearchCV yielded the most reliable results for MLP as it allowed targeted tuning and repeatable results.

This algorithm was chosen as it is reasonably flexible and suited to our data sets as it has well documented successes in research for classification and regression prediction on tabular data.

4 Experiment Methodology

All datasets were trained with all classifiers combining different settings and parameters. In order to be able to compare, improve and conduct conclusions there was the same methodology applied for all datasets. While the preprocessing and data preparation steps before training are more individual matters depending on the underlying dataset, the training and evaluation can be generalized across all datasets. Nevertheless, it should be noted that the modeling process is an iterative process were individual changes and loops have to be made every so often. According to the outcome of the results one has to follow the indications to receive a robust model that is able to generalize well. Since there are four diverse datasets, not the same parameters, settings or even classifiers give the same results. Still there are several patterns and steps that were followed uniformly to get to the best model taking specific possibilities into account for all classifiers and datasets.

4.1 Holdout

First, there was a simple holdout training done for each dataset without any parameter tuning. This means holding back an unseen validation dataset as a final sanity check for the model before it is used on the test set. Holding back unseen data is very important in order to avoid data leakage. Another way to deal with possible data leackage is adding random noise to the input data. However, this was not done since it would go beyond the scope of this exercise.
This first holdout training without hyper parameter tuning gives a first impression on the suitability of the classifier for the given classification problem and on the success of the preprocessing steps. This would be the first opportunity to go back and revise some preprocessing. It is worth getting a feeling for changes that can be achieved by different split rates and data manipulations. For example for the Congressional Voting dataset there were several arrangements taken based on these first results in addition to the earlier preprocessing. Missing values were replaced with -1 rather than with random values of 0 or 1. The column with the most missing values 'export-administration-act-south-africa' was kept in the dataset, whereas the 'ID' column was removed. Different splits of the training and test set gave different results across all classifiers from which no recommendated setting was able to be derived. That is why it was kept at 80/20 for the following trainings. For the Red Wine Quality dataset in contrast to that the holdout was the main strategy for splitting the data. All parameters, models and transformations were evaluated, trained and compared using this method. First a standard 80/20 split was deployed. After some experimentation a 90/10 split seemed to be more beneficial. This could be explained trough the additional data available for training. Although this data set is quite large, data for certain output classes is very sparse.

4.2 Parameter Tuning

A second step across all datasets and classifiers was to experiment with different parameter combinations and settings. This gave several results per classifier per dataset. First, it is worth playing with the different parameters manually in order to get a feeling for the changes in results. Nevertheless, it is quite difficult to find a good parameter tuning this way. A lot of times it seems like either a change of the same parameters does not have any impact on the results. At the same time most of the parameters seem to be quite sensitive for change, slingshotting the results in a worse direction of performance. Dealing with these settings is not very intuitive and hard to fully understand. For the Random Forest classifier the first attempt at parameter tuning was done using simple plots. One parameter was increased and the resulting accuracy plotted against the parameter value. This gave a rough estimate of the optimal values. Then, fine tuning was done using cross-validation on the transformed training set. However, this optimisation did not improve the prediction significantly. The noise was significantly higher than the improvement. The most important Random Forest parameter "n_estimators" seemed to perform well on surprisingly low values in the range of 10-30 for the Credit-Approval and Red Wine Quality data set, with no added benefit for higher values. The standard practice on the contrary is usually in the range of about 100-200.
Getting to the optimal model by just trying different hyper parameters was not possible here. Scikit-learn offers several automated searches for optimal parameter tuning, for example one called Gridsearch. Gridsearch was used for all datasets and classifiers in order to get some feedback on suitable parameter combinations. It should be noted that these suggested hyper parameter settings hold the risk of overfitting the data. This explained why several parameter suggestions with a promising high accuracy did not generalize on new data as well as the models without these tunings and a smaller accuracy.

4.3 Cross-Validation

Finally, the data was trained using cross-validation. The number of parts in which the data is split into equal sizes is easy to adjust and gives different results. Here there is a training-test as well, by which the performance is calculated, similar to the holdout. The difference is that the estimate of performance is given by the average (mean) of all validation set performances.
Cross-validation is easy to combine with grid-search. For the Red Wine Quality dataset every model was trained using 10-fold cross-validation. The resulting accuracy is comparable to the one achieved by holdout. Due to the averaging in cross-validation, results seemed to be more stable when rerunning the script with different seeds.
For the cross-validation of the Multilayer Perceptron a pipeline was used. These architectures in machine learning allow a sequence of multiple steps that do everything from data extraction and preprocessing to model training and deployment to be performed within cross validation folds. Pipelines are not only good to automate the workflow but are also another convenient way to avoid data leakage.

5 Model Evaluation and Comparison

5.1 Performance Measures

After the training of a model an important step is to measure its quality. All models need to be evaluated regarding their results in order to be able to improve them or examine how to use them. It is important to analyze whether a model has learned correctly, understand what the model has learned and investigate why a model has not learned correctly. For this we need expressive metrics and enable comparison between model performances on test data. There are many scores depending on how the classifier's behavior is aimed to be quantified. For this report we only chose metrics that are widely used. The following metrics are helpful to understand the model performances summarizing different conditions of interest. It should be noted that only a combination of them is sufficient enough in order to picture strengths and weaknesses as well as changes.

Confusion Matrix
The confusion matrix displays the number of instances per class that have been classified correctly and the number of falsely predicted instances with respect to their predicted classes. The true positives (TP) and true negatives (TN) are shown on the diagonal axis, where the ground truth equals the prediction. The rest of the matrix cells represent false predictions, either false positives (FP) or false negatives (FN). The colour of the cell usually indicates the greatness of the number it represents, the higher the number the darker the cell [4].
This visualization tool for the classifier accuracy shows very intuitively how good the model has learned, by indicating which classes the model is capable to identify and which classes it has difficulties to predict correctly. Furthermore, it is the basis of the following metrics that can be easily derived from the confusion matrix. This ensures a convention across all metrics that enables comparison.

Accuracy
Accuracy represents the quality of the classifier to predict all true positives and true negatives of a given dataset. It is calculated by $(TP + TN)/(TP + FP + TN + FN)$ [4]. Accuracy is suitable for our classification problems, where all class labels are equally relevant and our data is balanced. This is an important requirement, to avoid the accuracy being biased in favor of the majority class and conducting misleading interpretations since they do not take into account misclassification costs [2]. Furthermore, it is very easy to understand and intuitively to use. It covers a very basic information that is needed in order to evaluate the model performance. Nonetheless, it should be outlined that this measure alone is not sufficient as a metric for model evaluation.

Precision and Recall
Shortcomings of the accuracy require other scores that help to extend the evaluation [2]. Precision is a measure of how many of the predicted positives are true positives. It can be calculated by $TP/(TP + FP)$. Recall is a measure of how many true positives our model is able to find. It derives from $TP/(TP + FN)$ [4]. Precision and recall are a good complement of the accuracy metric, because it gives more information on the model performance. We are interested in how our model classifies specific parts of the data. Depending on the application it can be very important to

know about these performances. A bank for example is interested in a credit-approval with a high precision, so that as many of their clients as possible are able to pay pack their credits to them. Another reason why precision and recall should always be considered is that it makes it easier to measure and compare changes of model performances.

F1-score Another aspect the accuracy does not take into account is the trade-off between the quality of the classifier not only on positive but also negative data points. Such alternative score is for example the symmetric function called F1-score which calculates the harmonic mean of Precision and Recall and gives both the same weight:

$$F1 - score = \frac{2 * Recall * Precision}{(Recall + Precision)}[2]$$

.

References

[1] N Bu, T Tsuji, and O Fukuda. "EMG-controlled human-robot interfaces: A hybrid motion and task modeling approach". In: *Human Modelling for Bio-Inspired Robotics*. Elsevier, 2017, pp. 75–109.

[2] Theodoros Evgeniou and Massimiliano Pontil. "Support Vector Machines: Theory and Applications". In: vol. 2049. Jan. 2001, pp. 249–257. DOI: `10.1007/3-540-44673-7_12`.

[3] Rohith Gandhi. "Support vector machine—introduction to machine learning algorithms". In: *Towards Data Science* 7 (2018).

[4] Vijay Kotu and Bala Deshpande. "Chapter 8 - Model Evaluation". In: *Data Science (Second Edition)*. Ed. by Vijay Kotu and Bala Deshpande. Second Edition. Morgan Kaufmann, 2019, pp. 263–279. ISBN: 978-0-12-814761-0. DOI: `https://doi.org/10.1016/B978-0-12-814761-0.00008-3`. URL: `https://www.sciencedirect.com/science/article/pii/B9780128147610000083`.

[5] Leonardo Noriega. "Multilayer Perceptron Tutorial". en. In: (), p. 12.

[6] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[7] Rushikesh Pupale. "Support vector machines (SVM)—an overview". In: *A post at Towards data science available at https://towardsdatascience. com/https-medium-compupalerushikesh-svm-f4b42800e989* (2018).

[8] Hassan Ramchoun et al. "Multilayer Perceptron: Architecture Optimization and Training". en. In: *International Journal of Interactive Multimedia and Artificial Intelligence* 4.1 (2016), p. 26. ISSN: 1989-1660. DOI: `10.9781/ijimai.2016.415`. URL: `http://www.ijimai.org/journal/node/907` (visited on 11/14/2021).