Mandatory Assignment 01

Machine Learning F2023

Exercise 1 (Moons dataset)

Subtask A

When working with a Mulit-Layer Perceptron Classifier (MLPC) you have parameters such as hidden_layers and alpha. Hidden_layers specify the number of layers and nodes per layer that you would like to process your inputs. The alpha parameter specifies the regularization term, simply a value that changes the answer to be simpler. This can be used to prevent overfitting.

Specifically for the moons dataset, I found that using the parameters hidden_layers = (50, 25, 10), alpha = 0.01 works great as the input data is not restricted to too few nodes, which could lead to underfitting, yet you don't have too many to produce an output that is in the category of overfitting. The alpha parameter helps to prevent the output data from being overfitted, and thus working with a higher node count, a higher alpha value works great.

Subtask B

Standard logistic regression using the 'lbfgs' solver does not fit the moons dataset very well. This is more or so due to the curvature of two categories. However, using a polynomial logistic regression would probably be able to fit our dataset, however I feel that is out of the scope of this task.

The exact same goes for k-means. As we are dealing with two categories, setting a cluster size of 2 gives an output where the edge-data for each group is categorized wrongfully. This is once again due to the coordinate intercept of the two categories.

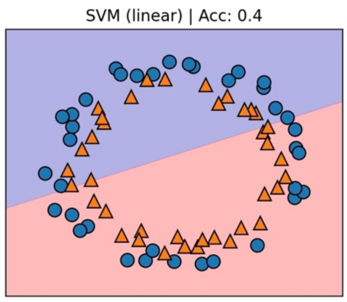
Decision trees work a little bit differently. When having a max_depth of 5, I feel you have a good balance between well fit and overfitted output. Overfitting is a big problem with decision trees and setting the max_depth of the tree can help. The solution I found makes a good categorization about 50% of the time. The same goes when using a random forest classifier.

When using a support vector machine (SVM), I found that having the regularization parameter set to around 100, is plenty to output more normalized predictions, while it not being too "simple". Using an SVM with the "RBF" kernel fits the moons dataset extremely well, while other kernels such as, "linear" or "sigmoid" don't fit the shape. An RBF SVM is the most efficient and effective choice when categorizing the moons dataset.

Subtask C

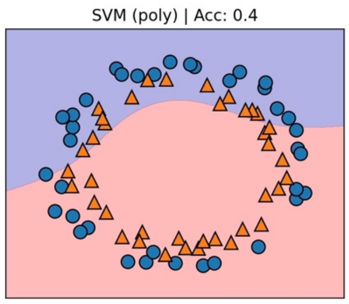
Looking at a couple different solutions using a Support Vector Machine Classifier (SVC) with various kernels, let's start with the linear kernel.

The linear kernel tries to find a common line that separates our two categories in our dataset (circles). However, it's clear to see that the linear kernel is not the best option for the circles-dataset. This is because our categories are mixed within each other and therefor basically impossible to find a straight line that separates the two. In this case, it does not matter what our other parameters for the SVC would be.

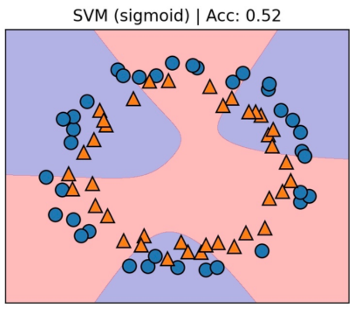


Another kernel we can use is the “poly” kernel. This is a kernel that tries to find a polynomial that separates our two categories.

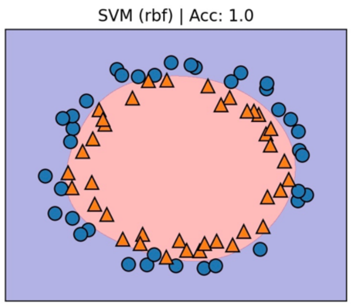
Looking back at the linear kernel, this is a step up in terms of categorizing our dataset correctly, but there is still much to be desired. While it seems the poly kernel can pick up more regularization between our two categories, it still has an accuracy score of 40%, which is the same as our linear kernel.



Because it is challenging to have positive parameters, the sigmoid kernel is typically troublesome or invalid. As it has a significant flaw—namely, that its output value range is not centered on zero—the sigmoid function is no longer frequently utilized in research. Here we can see that the sigmoid kernel does exactly that, it has a hard time classifying our categories, yet it scores better than the linear and poly kernels.



Looking at the Radial Basis Function (RBF) kernel, it is the exact kernel that we are looking for when it comes to the circles-dataset. The circles are namely scattered at the perimeter of two slightly different sized circles and therefore, the RBF kernel can perfectly classify decisions for the dataset. It must be said that even though we got an accuracy score of 100% this time, that is not the case for every single render. However, it is always in the 95th percentile.



Exercise 2

Subtask A

The titanic dataset is a classic example of a supervised learning problem. We are given data where the correct output for each entry is given, meaning that we can train our neural network (NN) in a supervised manner by weighing the NNs output to the correct output. The NN will get better and better at making correct decisions based on the input variables, as it is trained on the supervised training dataset.

We are given data with the following features (not including y-values): PassengerId, Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Cabin, Embarked. We will find that some of these features have a deciding factor in if the passenger ultimately does or doesn’t survive.

The y-values will consist of the following features: Survived. The passenger id feature is an identifier for the entry in the dataset (however, we will not be using this in the y-values), while the survived feature is the correct result.

Subtask B

Let’s consider what features we would like to keep for the training and test dataset. Some features have no effect on the result, while others do. It is important to remember that removing features from data, can be very dangerous as it’s easy to oversee some correlation from the feature to the result.

We can safely and without a doubt remove the features, passenger id, name, ticket, and fare. The passenger id has no relation to whether the person survives or not as it is just an identification for the entry. When looking at the name, you could argue that it could relate to the survival rate if you can denote their heritage or home country. However, I believe that is out of the scope of this assignment. The ticket and fare features are merely numbers that correlate to the persons wealth, but for this we will be using the class feature. Therefore, I have determined that these features are safe to remove from the dataset.

To talk a little bit about the features that we are keeping, Pclass or the class of the ticket could have an impact on the survival rate as the cabins could be closer to facilities, exits or they could even be treated differently to other class passengers. The sex of the passenger directly correlates to the potential strength, weight, ability to keep heat, who got on the emergency rafts, and much more. Age plays a role as older passengers could have decreased mobility. SibSp (# of siblings / spouses) and Parch (# of parents / children) could play a role in keeping each other safe and the risks that people take for their loved ones. Cabin determines how close the passenger is to certain facilities, exits and so on. And finally, embarked plays a role as the place that passengers have embarked from can play a factor in how rested the passenger is and so on.

When it comes to the training and test dataset, I have decided that for any missing data I will do the following:

- Passenger class → input the mean of all passenger classes (to try and skew the results as little as possible)
- Age → input the mean of all ages.
- Cabin → input '0' (as this feature is a string and we will be converting it later)
- Embarked → input 'S' (as this is the mode of all embarked entries)

We need to translate non-numeric data to numeric data, so for all features that include non-numeric data, I will do the following:

- Sex → input '0' for 'male' and '1' for 'female'
- Cabin → input the sum of the integer representation of the Unicode character for each character in the string.
- Embarked → input '0' for 'S', '1' for 'Q' and '2' for 'C'

Subtask C

I will be using a Multi-Layer Perceptron Classifier (MLPC) as the machine learning model for this dataset. We have used an MLPC on a narrower version of this dataset before, so I feel confident that I am able to implement an MLPC on the full size of this dataset. An MLPC is also one of the best models to use on a dataset like this as the many different inputs can be computed many, many times to produce different outputs. An MLPC is very suitable for a supervised learning set.

Under *assignment01/exercise2/c.py* you can find the code for training and testing the MLPC that I have implemented on the titanic dataset.

Subtask D

After several runs of the code, these are the best results I got using an MLPC.

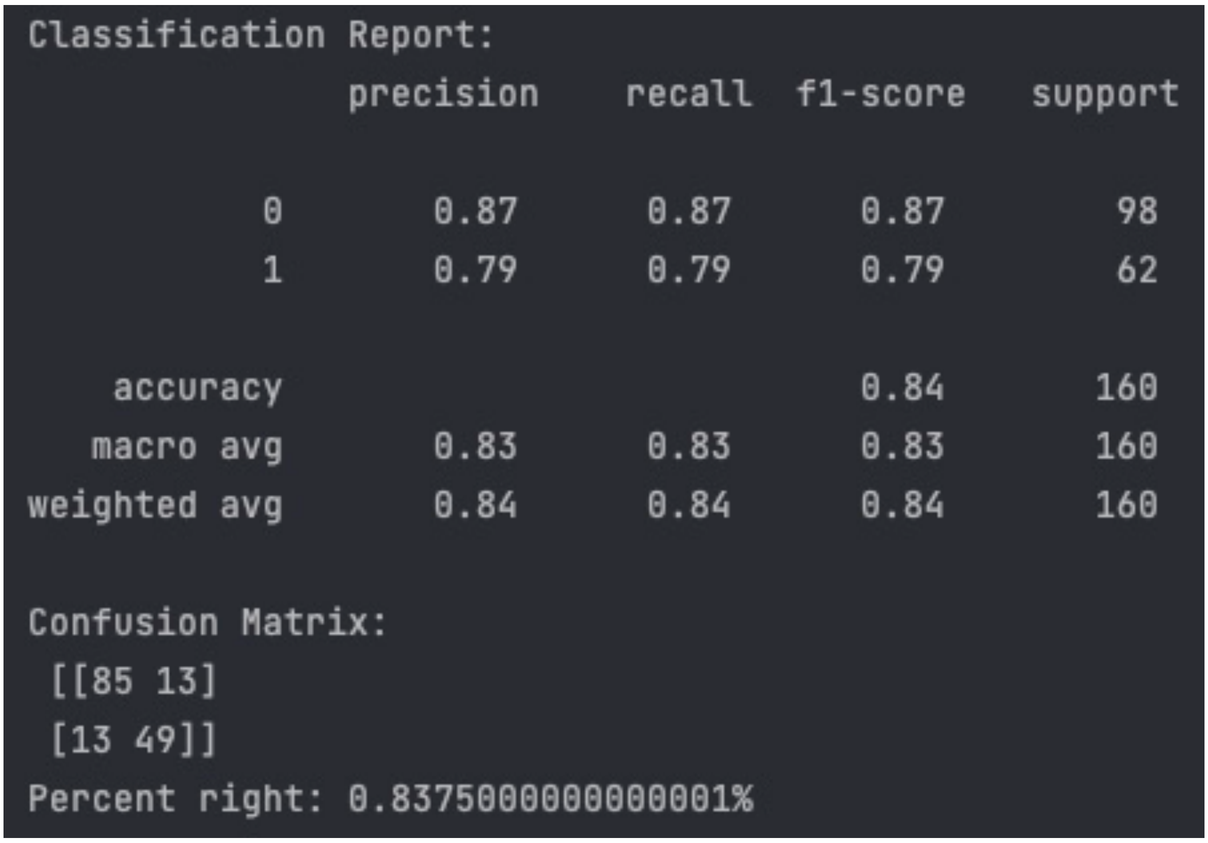
On the test I just ran, I got a precision of 85% for deaths and 84% for survivors. I got a recall rate of 91% for deaths and 74% for survivors.

In the code you can see the confusion matrix and I have used it to calculate that 84.4% of the outputs were in fact correct.

Looking at the confusion matrix, I can see that my model is worst at calculating survivors, at 65.2% correct, while deaths are at 89.9% correct.

Subtask E

I have decided to experiment with drawing the MLPC results up against a random forest classifier (RFC) and these are the best results I could get.



```
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.87      0.87        98
           1       0.79      0.79      0.79        62

    accuracy                           0.84       160
   macro avg       0.83      0.83      0.83       160
weighted avg       0.84      0.84      0.84       160

Confusion Matrix:
 [[85 13]
 [13 49]]
Percent right: 0.8375000000000001%
```

Looking at the results, I can deduce that the RFC is slightly worse at classifying whether a passenger has survived. It is however better at calculating deaths and worse with survivors then the MLPC, but not by much. Maybe trying out different input parameters could produce a better result for both models, but this is the best I could get after several tries.

Exercise 3

Subtask A

High-dimensional data can be made simpler using principal component analysis (PCA), while still preserving trends and patterns. It accomplishes this by condensing the data into fewer dimensions that serve as feature summaries. High-dimensional data are frequently found in biology and result from the measurement of various features, such as the expression of numerous genes, for each sample. When testing each feature for connection with an outcome, this type of data offers several difficulties that PCA mitigates, including processing cost and an increased error rate owing to multiple test correction. Like clustering, PCA is an unsupervised learning technique that identifies patterns without considering the samples' origins in various treatment groups or phenotypic variations.

PCA uses a small number of principle components (PCs) to geometrically project data onto lesser dimensions to discover the best way to summarize the data.

After having run calculation and plotted on a dimensionally reduced variant of the iris dataset, I can deduce that we can reasonable find at least 5 clusters. The unsupervised learning algorithm, k-means, has no problem finding many clusters, but looking at our “real classification” of the dataset, we can see that we are trying to find 3 clusters. Setting the cluster amount to 3 on the k-means algorithm, it does a good job at classifying the three clusters separately, but it does however confuse the labels 1 and 2. Therefore, we can see that the accuracy score is relatively low at about 45%, but flipping those labels gives us an accuracy score of 89%. To be honest, I have no clue why the k-means algorithm is confusing 1 with 2 and vice versa, but I am pleased with the accuracy score after flipping those labels.

Subtask B

PCA is good to use on larger datasets as removing features reduces the risk of overfitting. An algorithm can learn to understand any dataset, but overfitting causes the NN to produce false positives when new data is presented. On the other side, underfitting can have the same result, so it is important to not remove too many dimensions/features, but also having too many is just as bad.

When trying different settings for the Support Vector Classifier (SVC), I tried changing the kernel, regularization (C value), kernel coefficient (gamma) and random state with probability. I found that using the RBF kernel with a regularization of 500 and coefficient of 0.0001 gave me the best results, as high as 97% accuracy.

In terms of the PCA, I tried changing the number of components to a higher value, 300, and a lower value, 100. Using a higher value results in overfitted output data, as there are too many features for the SVC to classify. I could see this because the accuracy score went down to about 20%. Using a lower value, I presume resulted in underfitted data, as the accuracy was much less reliant between runs. Different entries would be classified incorrectly for each run. Therefore, I can denote that using a PCA with 150 features, is useful as it prevents overfitting and underfitting.

For another classification we could use another algorithm such as a Multi-Layer Perceptron Classifier (MLPC). This classifier, just as the SVC, is also a supervised machine learning algorithm so it would fit will with the Olivetti faces dataset.

If we want to look for one person, it would be rather simple. All we need to do is find the person we want to classify and save that person as our y_test data. Then when we are using our classifier to predict against our dimensionally reduces X test dataset, we want to check if the prediction is equal to our specific y_test data (the person we want to classify). You can check the file *assignment01/exercise3/b.py* to see this implemented in code.