Project - SF2935

Group 4
Magnus Axén maaxen@kth.se 980915-7119
Hannes Andersson hannes2@kth.se 990724-3217
Henrik Fagerlund hfag@kth.se 991012-2655
Jacob Holmberg jacobho@kth.se 980427-6377
John Sjöberg sjoberg8@kth.se 960710-6573

November 8, 2022

1 Problem statement and introduction

In the project we are given labeled data, comprising of roughly 500 data-points with various categorical attributes, e.g. mode, temp, energy, among others. The data-points have a given label, $\{0, 1\}$, indicating whether or not the user enjoys the song or not. We are asked in this project to try various statistical methods to train a binary classification algorithm for predicting new labels for a test data set.

2 Methods

For this project a few different methods have been used to approach the problem. Below are the descriptions of the chosen methods.

2.1 Neural Network

One of the methods to categorize data without a clear pattern is to implement a neural network. A neural network is good to use as it is able to find complex patterns. However this also comes with the cost of having a complex model. For this implementation a *feed forward network* has been chosen and been implemented with the pre-made python package Keras.

A feed forward network consists of the input layer, hidden layer(s) and an output layer. The layers are built up with nodes that are linear combinations of the previous layer brought through an activation function, see figure below:

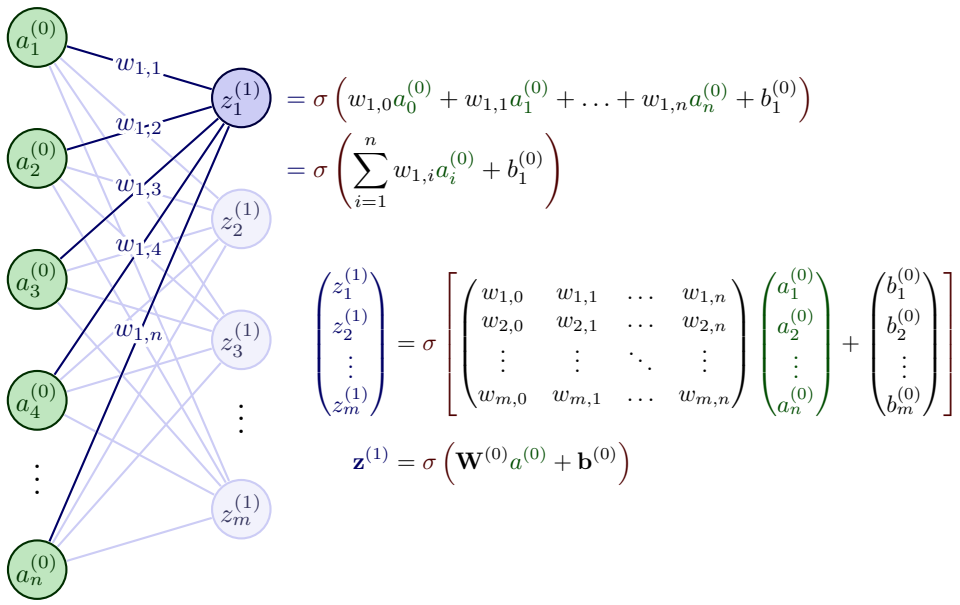

Figure 1: Showcasing conceptual image of nodes and layers in neural network.

In figure 1 the σ is the activation function where $\sigma(x)$ is typically a Sigmoid function or similar. Z is the assigned value to the node where $Z_i^{(L)}$ describes node i in layer L, $i \in [1, M_L]$. Figure 1 also shows that values of nodes in layer L, $Z_i^{(L)}$, are dependent on the values of the nodes in the previous layer $Z_i^{(L-1)}$, through the weights and activation function. Apart from that, it also shows how the previous layer communicates to the next layer and this is the principle of a feed forward network. The input layer and its nodes are transformed into the next layer and so on until the output layer is reached. How one determines the best amount of layers and number of nodes in each layer is not clear and one needs to resolve to intuition and trial-and-error here.

The activation function is chosen subjectively and worth mentioning here is that if the activation function is a linear mapping, the output is going to end up as being linear combinations of the input. So the activation function is therefore what enables the neural network to get more complex decision boundaries. Once the activation function and weights are determined the network can process the input and give an output, however the weights have to be decided for the transforms to work. The weights cannot be solved analytically and we hence need to "learn" the weights by using the training data, a loss function dependant on the weights, $L(\mathbf{w})$, to optimize and some form of gradient method.

The way to estimate the weights is to use the errors received in the output layer, i.e $\delta^{L+1} = \hat{y_j} - y_j$ where $\hat{y_j}$ is the estimation in the output layer, and then propagate it backwards into the neural network by using the backpropagation formula:

$$\delta_j^{(m)} = \sigma'(a_j^{(m)}) \sum_k w_{k,j}^{(m+1)} \delta_k^{(m+1)}.$$

By using this and the fact that it is possible to express $\frac{\partial L_n}{\partial w_{j,i}^{(m)}}$ as:

$$\frac{\partial L_n}{\partial w_{j,i}^{(m)}} = \delta_j^m Z_i^{(m-1)},$$

it is thus possible to estimate the derivatives of the weights and therefore also possible use a gradient based method to improve the weights. Though it should be mentioned here that there is no guarantee that the weight-space have a convex behavior so there is no guarantee for an optimal solution, just a better solution to the loss function than the initial guess of the weights.

In learning we need to form a suitable Loss function which can then be optimised over so that the network can update its weights. Seeing as our problem formulation is to predict a binary label we need to consider a suitable Loss function for this end. Hence the Cross-entropy loss function is used, i.e.

$$L(w) = -\sum_{n=1}^{N} (y_n \log(\hat{y}(x_n, w)) + (1 - y_n) \log(1 - \hat{y_n}(x_n, w)).$$

Here it can be seen when the prediction $\hat{y}(x_n, w)$ and y are the close to one another (when $y = \{0, 1\}$) the loss function becomes small, and if it predicts wrong vice versa.

Seeing as there is no clear analytical solution to the problem, $\nabla_w L(w) = 0$ we instead form an iterative process, updating the weights via the training data. This via the formula

$$w^{\tau+1} = w^{\tau} + \Delta w^{\tau},$$

where τ is reflective of the number of iterations. The choice of updating methods vary, we selected for Adams method, that is

$$w_{t+1} = w_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}} + \epsilon}.$$

Without going into to much details the $\hat{m}_t$ and $\hat{v}$ are based of the various moments of the gradients and epsilon some small constant to avoid division by 0 [1]. More conventional methods are Gradient descent or Stochastic gradient descent. Moreover, η is the learning rate of the method, which is a hyper-parameter which should be optimised. As if it's too low the loss function wont improve and if set too high it does diverges. This can be optimised by looking at a range (using Cross-validation) and inspecting their respective loss values.

2.2 Random forest

The second method that was used is a random forest classifier which consists of several decision trees. A decision tree is a good classifier when the data is a mixture of continuous and categorical variables since the node impurity function only determines which feature to split by handling each feature separately and then compare them.

One problem is that decision trees may have high variance since it is very dependent on the training data which can lead to over-fitting, therefore one can use a random forest which reduces the variance. A random forest is made up by using bagging combined with training each tree without using all available features in a random manner. I.e., for every bootstrap sample before each split, we choose a fixed number of features $m \leq p$, where p is the total number of features, uniformly which prevents the training data for each tree from being correlated. When using a random forest for classification which is the case for this project, the output is the most frequent output of all decision trees in the random forest.

2.3 KNN

The third and last method used is the K-Nearest Neighbors (KNN) which is one of the more simple and easy-to-implement non-parametric machine learning method. The basics of the method is the assumption that similar things exists in close proximity, i.e. elements belonging to the same class are near to each other.

KNN measures the distance between the data point under investigation and all the given data points, groups the K (a chosen value) nearest neighbors, and assigning the data point to the class that arises most frequently in this subgroup, using so called majority vote.

The errors given by the method can differ vastly depending on the choice of K. Therefore, choosing the right value for K is of big importance when using this method. By iterating the method over different K-values one can investigate the root-mean-square error (RMSE) over the different models in order to find the K-value to give the smallest RMSE. This will correspond to the "best" possible KNN-model.

3 Application of models on data

On a general note, we identified some points that were outside the range that was given in the project description, i.e. energy being in the interval $[0, 1]$ and loudness $[0, -60]$. These points were removed.

3.1 Neural Network

As a Neural Network in this instance will use a combination of linear models to classify the label we decided to adjust and remove the labeled categorical data. More precisely we tried the one hot encoding, however this led to the data being more volatile in the testing, which we think could be as a result of the larger network size and trouble in finding an appropriate weight for the single $\{0, 1\}$ values. The remainder of the features were used in the training, as we deemed them unique to each label (see histogram in appendix).

3.2 Random forest

As mentioned earlier the decision tree only compares the separately computed impurities for each feature and therefore there is no need for normalisation of the data. This is since the splits for each region of the continuous variables will split the same proportion of data so normalising would only lead to unnecessary computations in our case.

The hyper parameters that was chosen manually in the random forest was the number of estimators (trees), the maximum depth of each tree and the number of features m used in each tree. Furthermore, the measuring criterion of the node impurity was done with cross-entropy. The number of trees, n , can be chosen quite high without adding any variance due to the aggregating step in the learning process and was chosen as $n = 100$ since it is standard. A maximum depth of each tree was chosen as a deeper tree tends to overfitting of the training data which might generalize bad on new data. The number of features, m, for each tree was chosen as it prevents the trees from becoming highly correlated with the requirement that $m < 11$,i.e. smaller than the total number of features or it would only be standard bagging.

To determine the value of the hyperparameters, a grid search with cross-validation was used. The grid search works such that given a set of possible values for each parameter, a model is trained with each combination of those parameter values and tested against a validation set where one can obtain a measure of the accuracy of the model. This measure can sometime be missleading so by adding cross-validation one can get a better measure of the error. The cross-validation works by just training more models for each parameter combination and take the average of the error from those models. In this task a 5-fold cross-validation was used since to many folds will lead to few training samples.

The result from the grid search is presented in table 1.

Table 1: The best parameters obtained from the grid search.

Hyperparameter	Best Value
`criterion`	entropy
`max_depth`	10
`max_features`	7

3.3 KNN

The K-value obtained when iterating and comparing RMSE on the data varied between 25 to 30, further testing showed that using $K = 29$ often resulted in small RMSE and is therefore the chosen

K-value. The model can also be built using GridSearchCV. This is a package that is using cross-validation to find the optimal values for given sets of hyperparameters. This gave a bigger variation of K-values since it tries to fit the model as well as possible. When iterating both using $K = 29$ and GridSearchCV, it was discovered that using $K = 29$ gave a better average result. The type of distance used for this method is euclidean distance.

4 Model evaluation

In validating the models we did a 80/20 train/test partitioning for estimating the models accuracy. The average scores on the validation set for each method is presented in table 2.

Table 2: This table shows the accuracy's for the different methods.

Method	**Accuracy**
RF	0.86
NN	0.80
KNN	0.68

Since the training data exists of relatively few data points it can be hard to trust the results of a neural network as it often demands a larger quantity of data to be reliable. Since the data also contains categorical features the neural network is once again not preferred as it is better for continuous data. Since the random forest method performed better than KNN on the training set, the model from random forest has been chosen to use for the final decision making.

5 Result

From the Random forest we got a 70 percentage accuracy as an initial submission.

6 Conclusion

Since the data is a mixture of categorical and continuous features it is not surprising that the random forest performed well on the data set. It also perform well on data sets with few data points compared to the neural network which needs a large amount of data to perform.

The same applies to KNN which also performs fairly well on smaller data sets. It is also easy to implement with only one hyperparameter. The challenging part about KNN is that it responds poorly on imbalanced data, i.e. when there is a majority of one class in the training data, and when the number of input variables grows, i.e. curse of dimensionality. This could be reasons for it to work poorly on the data set.

7 Code

https://github.com/Axenmagnus/SF2935

8 Appendix

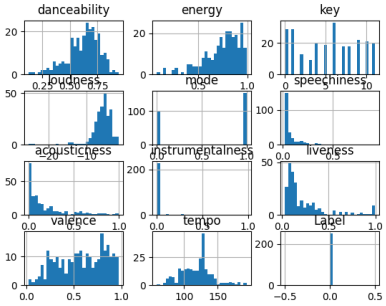

Figure 2: Histogram Label 0

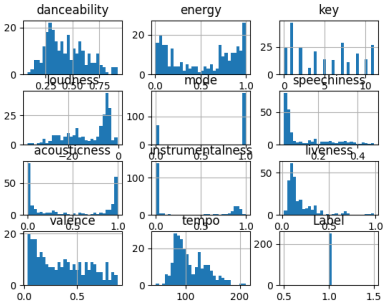

Figure 3: Histogram Label 0

References

[1] Akash Ajagekar, https://optimization.cbe.cornell.edu/index.php?title=Adam