SF2935
Spotify Songs Classification
Group 020

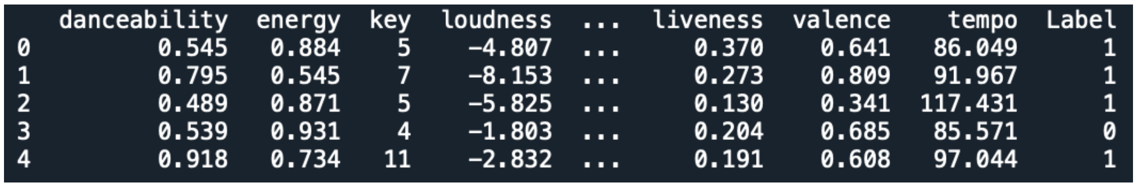
Yonas Melake yonasm@kth.se
Leonardo Nogarin nogarin@kth.se
Malika Satayeva satayeva@kth.se
Shriya Bhatija bhatija@kth.se
Raghav Singhal raghavsi@kth.se

November 8, 2022

1 Problem Overview

The aim of the project is to use machine learning methods from the course SF2935 to train a model that predicts [redacted] song preferences. The data available consists of:

- a training set of 505 songs labelled as 1 (”liked”) or 0 (”disliked”) and 11 features, both discrete and continuous, as stated in Spotify’s web-API
- A test set of 75 songs in which the same features are listed but labels are not available.



```
   danceability  energy  key  loudness  ...  liveness  valence    tempo  Label
0         0.545   0.884    5    -4.807  ...     0.370    0.641   86.049      1
1         0.795   0.545    7    -8.153  ...     0.273    0.809   91.967      1
2         0.489   0.871    5    -5.825  ...     0.130    0.341  117.431      1
3         0.539   0.931    4    -1.803  ...     0.204    0.685   85.571      0
4         0.918   0.734   11    -2.832  ...     0.191    0.608   97.044      1
```

Figure 1: A snapshot of the training data

The objective is to assess the predictive power of such models and select the best one in order to predict the labels of the songs in the test set.

Before starting to train models, by simple inspection of the data, we noticed that 3 data points in the training data set were showing unusual features compared to the rest. Namely, the ones corresponding to rows 43,84,94 (following a Python enumeration starting at 0). Row 43 is the only one to have a positive value for the loudness, line 84 has an out of range energy ($O(10^3)$ while the rest of the data are $O(10^{-1})$, and finally, line 94 has a loudness of the order of 10^3 in absolute value while the rest of the data are $O(10^1)$ at most.

We agreed that these anomalies may be caused by mistakes in the collection of data, simple typos perhaps, so we tried training models both omitting and keeping such data points. The result was that, in general, all models were better at prediction in the first case. Hence, we decided to go without such data points in the method we chose to put in production.

2 Support Vector Machines

2.1 Overview on SVM

SVM is the kernelised version of the support vector classifier, which in turn is a generalisation of the maximal margin classifier in the case of a non-linearly-separable feature space. In the context of linear binary classifiers, this first method relies on maximizing the distance between the hyperplane dividing the data points in the two classes and the elements of such classes.
Back to maximal margin classifier, finding the parameters that define the hyperplane separating the two classes involves solving the following (primal) optimisation problem:

$$\begin{aligned} &\min_{\omega_0,\omega} \frac{1}{2}||\omega||^2 + C\sum_{i=1}^{n} \xi_i \\ &s.t.\ y_i(\langle \omega, x_i\rangle + \omega_0) \geq 1 - \xi_i,\ \xi_i \geq 0,\ i = 1, ..., n \end{aligned} \tag{1}$$

The name support vectors comes from the fact that only some data are significant in the training of the method, namely the ones corresponding to $\xi \leq 1$.

To extend this method to be non linear, we make use of kernels and obtain SVM.

2.2 Implementation of SVM and model assessment

In order to use support vector machines (SVM) for classifying the songs from the playlist we proceeded in the following way.

First of all we normalized all the continuous predictors, mapping them to the interval $[0, 1]$. The normalization was carried up on both the training and test sets together.

We then proceeded to test SVM on the training data with different kernel choices, namely linear (i.e. standard SVM), polynomial, Gaussian and sigmoid. Python offers the function ”SVC” in the module ”sklearn.svm”, which fits an SVM classifier on the training data and requires to specify the chosen kernel.

first of all, for each kernel choice we fitted the model on the whole training set and assessed the training error. The results are provided in the table below.

Kernel Choice	Training Score (whole train data set)	Training Score (excluding anomalies)
Linear	0.794	0.795
Gaussian	0.832	0.851
Polynomial (degree 12)	0.99	1.0
Sigmoid	0.529	0.501

From the table we can observe that sigmoid kernel is performing very poorly and for this reason is to be discarded from further assessments. At the same time, standard SVM and Gaussian perform similarly well, with the latter doing slightly better. Finally, provided that the degree of the polynomial is high enough, polynomial kernels show a training error that is arbitrarily small. However, such a model is probably overfitting the data, as the next results will show.

In order to test the predictive performances of SVM for each kernel choice, we randomly split the training set, selecting 80% of the data for training while testing on the remaining 20%. We ran this experiments multiple times and found, as expected, that the polynomial kernel SVM was performing much worse at prediction than the training error would suggest. On the other hand, linear and Gaussian choices were consistent with the previous results. The table below shows the test scores for linear, Gaussian and Polynomial (degree 10) kernels over 10 trials.

Kernel Choice	Average Test Score (whole train set)	Average Test Score (excluding anomalies)
Linear	0.768	0.771
Gaussian	0.799	0.822
Polynomial (degree 12)	0.738	0.756

2.3 Final Remarks

In conclusion, our analysis showed that Gaussian was the best kernel choice for predicting song labels, just slightly better than standard SVD. In addition to this, excluding the anomalies in the data slightly improves the prediction scores on average. Nonetheless, such improvement is minimal and has little statistical relevance for this class of methods.

3 Linear and Quadratic Discriminant Analysis

Linear- (LDA) and quadratic discriminant analysis (QDA) comes from the approach of modelling the distribution of the predictors X in each class and then using the following equation from Bayes' theorem

$$\mathbb{P}(Y \in C_k | X = x) = \frac{p_{X|Y}(x|C_k)p_Y(C_k)}{\sum_y p_{X|Y}(x|C_k)p_Y(y)} \tag{2}$$

, where $p_{X|Y}(\cdot|C_k)$ denotes the conditional density of X given $Y \in C_k$ and p_Y is the prior for Y, which is estimated using the training data $\tau = \{(x_i, y_i)\}_{i=1}^n$ according to

$$\hat{p}_Y(y) = \frac{1}{n}\sum_{i=1}^n I\{y_i = y\}. \tag{3}$$

In LDA and QDA the conditional densities are assumed to be multivariate Gaussian,

$$p_{X|Y}(x|y) = \frac{1}{(2\pi)^{p/2}|\Sigma_y|^{1/2}} e^{-\frac{1}{2}(x-\mu_y)^T\Sigma_y^{-1}(x-\mu_y)} \tag{4}$$

, where $x \in \mathbb{R}^p$. Thus, the decision boundary for two classes $y, \tilde{y}$ is

$$\{x : \mathbb{P}(Y = y|X = x) = \mathbb{P}(Y = \tilde{y}|X = x)\} \tag{5}$$

, which gives the following equation for QDA

$$\log\frac{\mathbb{P}(Y = \tilde{y}|X = x)}{\mathbb{P}(Y = y|X = x)} = \log\frac{p_{X|Y}(x|\tilde{y})}{p_{X|Y}(x|y)} + \log\frac{p_Y(\tilde{y})}{p_Y(y)} = 0$$

$$\Rightarrow \log\frac{p_Y(\tilde{y})}{p_Y(y)} + \log\frac{(2\pi)^{p/2}|\Sigma_y|^{1/2}}{(2\pi)^{p/2}|\Sigma_{\tilde{y}}|^{1/2}} - \frac{1}{2}(x-\mu_{\tilde{y}})^T\Sigma_{\tilde{y}}^{-1}(x-\mu_{\tilde{y}}) + \frac{1}{2}(x-\mu_y)^T\Sigma_y^{-1}(x-\mu_y) = 0. \tag{6}$$

In LDA, it is assumed that $\Sigma_y = \Sigma_{\tilde{y}} = \Sigma$, thus the corresponding equation is

$$\log\frac{p_Y(\tilde{y})}{p_Y(y)} + \log -\frac{1}{2}(\mu_{\tilde{y}} + mu_y)^T\Sigma^{-1}(\mu_{\tilde{y}} - \mu_y) + x^T\Sigma^{-1}(\mu_{\tilde{y}} - \mu_y) = 0. \tag{7}$$

3.1 Implementation

First, to find the best predictors of the variables, a Generalized linear model (GLM) model with all the variables was created using this code line in R Code:

$$glm.model < - glm(Label ., data = data, family = binomial(link =' logit')).$$

The following code line was used to find out how well a given variable is improving the model:

$$summary(glm.model)$$

, which produced the output shown in Listing 1. The output was interpreted that the vaiables danceability, energy, speechiness, liveness and tempo are the significant predictors. Thus, LDA and QDA models were created using those variables and all variables.

Listing 1: Output about variable performance

```
Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)       5.50350    1.57911   3.485 0.000492 ***
danceability     -5.44215    0.91333  -5.959 2.54e-09 ***
energy           -2.32185    1.13975  -2.037 0.041635 *
key              -0.03940    0.03323  -1.186 0.235762
loudness         -0.62076    2.02124  -0.307 0.758752
mode              0.50962    0.27184   1.875 0.060829 .
speechiness       8.80931    1.21656   7.241 4.45e-13 ***
acousticness     -0.96791    0.71217  -1.359 0.174117
instrumentalness  1.34110    0.69279   1.936 0.052892 .
liveness         -2.26608    0.61527  -3.683 0.000230 ***
valence          -0.47235    0.59707  -0.791 0.428878
tempo            -1.76870    0.79358  -2.229 0.025830 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The models were tested by randomly selecting 75 data point as test data and using the rest as training data. The figures 2 and 3 show the histograms of each models accuracy rate from 1000 tests, see 5-fold cross-validation scores of the models in table 1. Their is no significant difference between the models' performance, but the QDA model with only the significant variables is the best performing. But, QDA models were very sensitive to data points with abnormal values, because when the data points at rows 85 and 95 (R indexing) were included their average accuracy rate falls to round 70%. While the LDA models performed roughly as well.

Table 1: Test scores for LDA and QDA models

Model	**Test scores**	**Standard deviation**
LDA with all variables	0.793	0.0441
LDA with only significant variables	0.785	0.0557
QDA with all variables	0.787	0.0555
QDA with only significant variables	0.811	0.0576

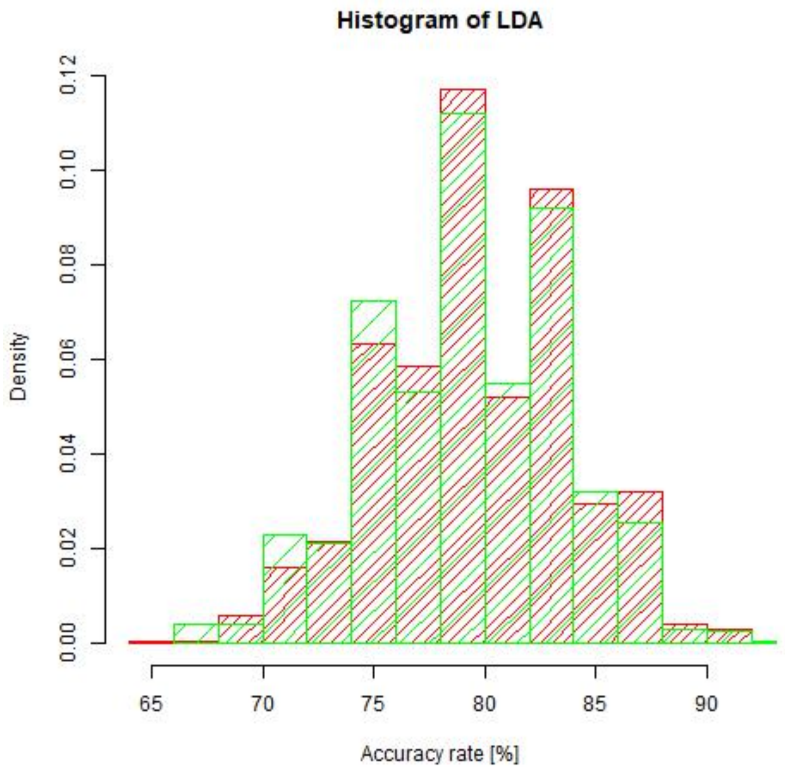


Figure 2: Histogram of the accuracy rate of LDA, with all the variables (red) and only the significant variables (green)

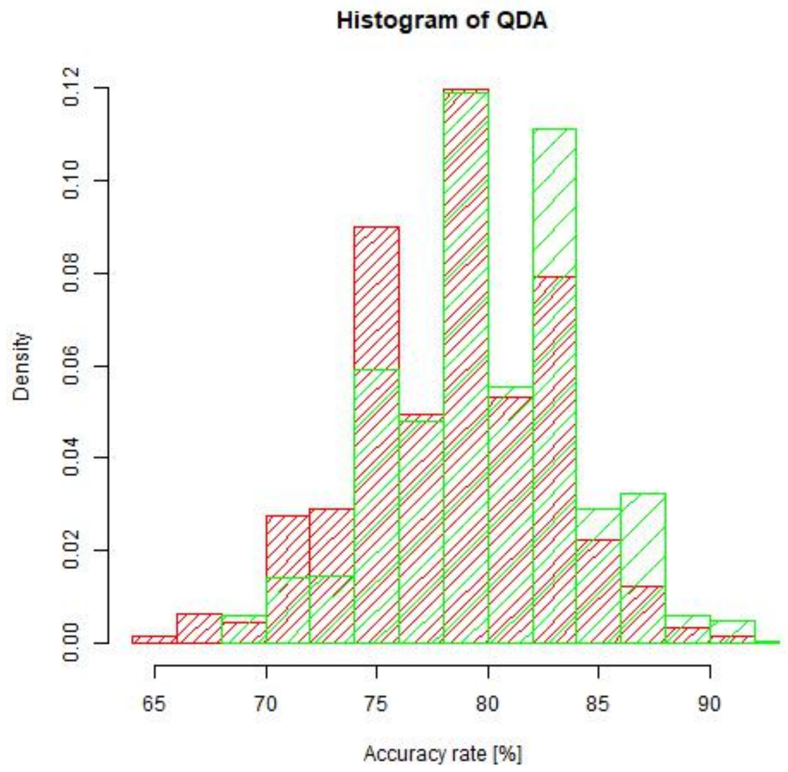


Figure 3: Histogram of the accuracy rate of QDA, with all the variables (red) and only the significant variables (green)

4 Classification tree

Given the structure of the data set: 9 continuous and 2 discrete features, a classification tree seems like a natural choice to consider. Here we used a `sklearn` implementation of the decision tree classifier. The idea of the classification tree is to construct an optimal partitioning $R_1, \dots, R_m$ of the input feature space $\mathcal{X}$ so that in each domain we can use simpler classification rules to predict unseen data, such as the majority of the training sample votes in that domain. Each domain in the partitioning is represented by a terminal node in the tree, and we aim for the terminal nodes to be as homogeneous as possible (ideally, pure). The most common measures of node impurity are Gini index and Cross-entropy (both are differentiable functions), so a feature with a cut-point value minimizing the measure of impurity of child nodes is used for splitting the parent node.
To begin with, we grew the tree with default `sklearn` settings such as no tree depth limit, measure of impurity is Gini, and the minimum number of samples that can reside in the terminal nodes is 1. If we let the tree grow to full depth, which for our data is 12, the training error will reach the minimum possible value of 0, and we will get an overfitted model. To regain control over tree complexity, we follow a common post-pruning strategy, which is weakest link pruning with $\alpha \geq 0$ as the regularization parameter. The goal is to collapse the nodes that result in the smallest increase in the cost function $\mathcal{L}$, which is defined through the Gini index (default `sklearn` setting) Q_m of each terminal node m.

$$\mathcal{L}(T) = \sum_{m=1}^{|T|} N_m Q_m$$

where N_m is the number of training examples in region R_m, and $|T|$ is the number of terminal nodes in the tree T.

$$Q_m = \sum_{k=1}^{2} p_{m,k}(1 - p_{m,k})$$

where $p_{m,k}$ is the probability of being classified with label k at terminal node m.
Therefore, pruning forces us to trade bias versus variance by setting positive α. We first compute a sequence of effective α's that are effective in the sense that the difference in cost-complexity C_α of the tree T and the pruned tree T_{pruned} is 0.

$$C_\alpha(T) = \mathcal{L}(T) + \alpha \cdot |T|$$

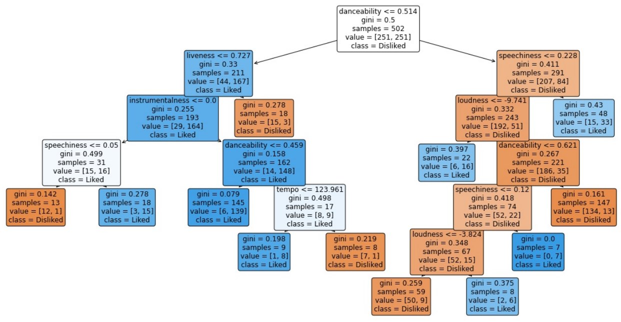


Figure 4: Pruned decision tree, $\alpha = 0.007$

The reason is that there is a 1-1 correspondence between a sequence of effective α's and a sequence of weakest link nodes. Based on a 5-fold stratified cross-validation average score, the final α is selected. It turns out to be 0.007, and the resulting tree looks like the one shown in figure 4. Its performance can be summarized as follows: train accuracy 0.88, mean test 5-fold cross-validation score 0.767 with a standard deviation of 0.102.

5 Neural Networks

5.1 Introduction

In this section, we used the method of neural networks to train the classifier. We implemented the same using modules in the sklearn library. We used an 80-20 training-validation split with the data.

5.2 Initial Processing

We initially normalized the data to a range between 0 and 1 using the MinMaxScaler module. We did this to improve performance by keeping all the inputs within the same range. The same transform was applied to the test data as well.

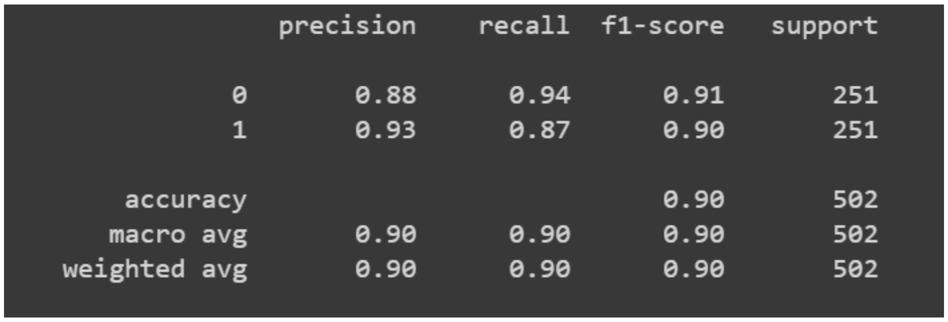
5.3 Implementation

The classification itself was done using a shallow neural network. The network consists of 3 hidden layers with sizes 12, 12, and 8, respectively. For this, we used an implementation of multi-layer perceptron from `sklearn` called MLPClassifier. The inputs consist of 11 features, and the output has 2 options - Like or Dislike. The activation function used was the Rectified Linear Unit (ReLU). The choice was motivated by the fact that it is a very traditional choice in neural networks, and is used to introduce non-linearity into the system. The optimizer used was Adam. Adam is a stochastic gradient-based optimizer algorithm that is among the state-of-the-art in training neural networks. The initial learning rate by default is 0.001. The value of epsilon for the numerical stability of Adam is 1e-8. MLPClassifier implements L2 regularization by default, with the default value of the α parameter as 0.0001. The batch size used was the default batch size of 200. The loss used is the Categorical Cross Entropy Loss. We trained the neural network for 2000 iterations to allow the loss to converge.

5.4 Results

We then printed out the confusion matrix and the classification report. The precision, recall, f1-score, and support, along with the accuracy, were noted. We finally made the predictions on the normalized test data.

To have a uniform metric for all the methods, we also evaluated the 5-fold cross-validation accuracy.



```
              precision    recall  f1-score   support

           0       0.88      0.94      0.91       251
           1       0.93      0.87      0.90       251

    accuracy                           0.90       502
   macro avg       0.90      0.90      0.90       502
weighted avg       0.90      0.90      0.90       502
```

Figure 5: Classification Report

The mean of the 5-fold validation accuracy was 70.36%.

6 Logistic Regression

6.1 Overview on Logistic Regression

Logistic regression is a parametric method for (binary) classification that produces a linear decision boundary in the respective input space. It relies on modelling the conditional distribution of a label $y \in \{0,1\}$ given an input $x \in \mathcal{X}$, which enables us to incorporate a measure of uncertainty into our model. A classification decision is made by evaluating the conditional probabilities of $y = 0|x$ and $y = 1|x$. More precisely, this is done in the following way:

Let $\mathcal{T} = \{(x_i, y_i)\}_{i=1}^{n}$ with $x_i \in \mathbb{R}^p$ and $y_i \in \{0,1\}$ be our training data. The general approach is to use the following assumption for the conditional probabilities:

$$\mathbb{P}(y = 1|x) = \frac{e^{w^T x + w_0}}{1 + e^{w^T x + w_0}} = \sigma(w^T x + w_0) \tag{8}$$

$$\mathbb{P}(y = 0|x) = 1 - \mathbb{P}(y = 1|x) = \frac{1}{1 + e^{w^T x + w_0}} = \sigma(-w^T x - w_0) \tag{9}$$

where $w \in \mathbb{R}^p, w_0 \in \mathbb{R}$ are the model parameters.

The classification rule associated to this model is the following: An input $x \in \mathbb{R}^p$ is assigned to the label $y = 1$ iff $\mathbb{P}(y = 1|x) > \mathbb{P}(y = 0|x)$. Hence, the decision boundary consists of $x \in \mathbb{R}^p$ for which it holds $\mathbb{P}(y = 1|x) = \mathbb{P}(y = 0|x)$ or equivalently $w^T x + w_0 = 0$, i.e. $\mathcal{D} = \{x \in \mathbb{R}^p | w^T x + w_0 = 0\}$.

The learning task in logistic regression consists of finding parameters w, w_0 that best explain the training data in terms of the above mentioned classification rule.

6.2 Implementation and model assessment

In order to apply Logistic Regression for the spotify song classification task we use the Logistic Regression classifier from the scikit-learn library in Python. This classifier implements regularized logistic regression with the L2 penalty term. The penalty aims to avoid overfitting.

Firstly, we load the data and remove the rows 43, 84 and 94 (python enumeration) as it is explained in the first section. Next, we extract the feature matrix to which we apply scaling such that all features lie within the range of [-1,1]. Now our data is in a suitable format for further implementations.

We define a function for 5-fold cross validation which we want to apply to evaluate our model with respect to the following measurements: training Accuracy scores, mean Training Accuracy, validation Accuracy scores and mean validation Accuracy. Particularly, we are interested in the mean validation accuracy which will indicate how the model performs on unseen data.

Finally, we train our logistic regression model from scikit-learn using 5-fold cross validation. The accuracy scores during cross validation are the following:

$$\textbf{Training accuracy scores:} \quad [0.87032419, 0.81546135, 0.77114428, 0.78606965, 0.80597015] \tag{10}$$

$$\textbf{Validation accuracy scores:} \quad [0.6039604, 0.7029703, 0.89, 0.78, 0.76]. \tag{11}$$

We further observe that the average training accuracy is 0.809 while the average validation accuracy is slightly lower at 0.747.

7 Conclusion

With all 5 methods in mind, we attempt to choose the best subset of models based on the 5-fold cross-validation scores tabulated in 2. Looking at Table 2, we first assume that all the QDA and LDA forms considered as well as Gaussian and Linear SVM's may (with no guarantee) generalise better than the rest of the methods. However, by trial and error (several submission attempts during competition) we learned that test label predictions made by above mentioned methods are less accurate than those made by pruned decision tree. One possible explanation could be that we're always working with specific instances from the distribution generating the data, and so there is a risk that training set does not properly cover the part of the features space with the test examples. As a result, unexpected perfomance on unseen data. However, perhaps, not only 'luck' underlies the dominant effectiveness of a decision tree (pruned) on the test data (77.3%), they are also known to work well on mixed data type, which was the case for us. In addition, they are easy to interpret, see figure 4. Overall, we've selected a pruned classification tree.

Method	**CV average**
Linear SVM (SVC)	0.781
Gaussian SVM	0.802
Polynomial (12) SVM	0.759
LDA (all features)	0.793
LDA (significant only)	0.785
QDA (all features)	0.787
QDA (significant only)	0.811
Pruned tree	0.767
NN	0.704
Logistic regression	0.747

Table 2: 5-fold cross-validation (CV) results