Machine Learning BSMALEA1KU

Machine Learning Exam

Forensic Identification of Glass Fragments

Frederik Bechmann Faarup - frfa@itu.dk
Frederik Peter Høngaard - frph@itu.dk
Kasper Thorhauge Grønbek - kgro@itu.dk

IT University of Copenhagen

Autumn 2020

Contents

1 Introduction

This project is an exercise in identification of glass fragments for forensic analysis. The report is subdivided into three major parts; the first part considers data visualization and dimensionality reduction, the second part introduces the classification models, our application of the models and discusses our expectations thereof. The last part of the paper discusses the performance of each model and some additional considerations for the perspectives of this classification task.

2 Data Visualization

Before applying any classification model on the data, it is desirable to get some visual representation if possible. Unfortunately there is no meaningful way of representing 9-dimensional data, so we must apply a method of dimensionality reduction to our data.

In the below figure, we can evaluate new components obtained by applying **L**inear **D**iscriminant **A**nalysis (LDA) and **P**rincipal **C**omponent **A**nalysis (PCA) as described in [1] pages 187 and 561. The former method aims to project the data into components that maximise class separability while the latter projects into components with directions of maximum variance.

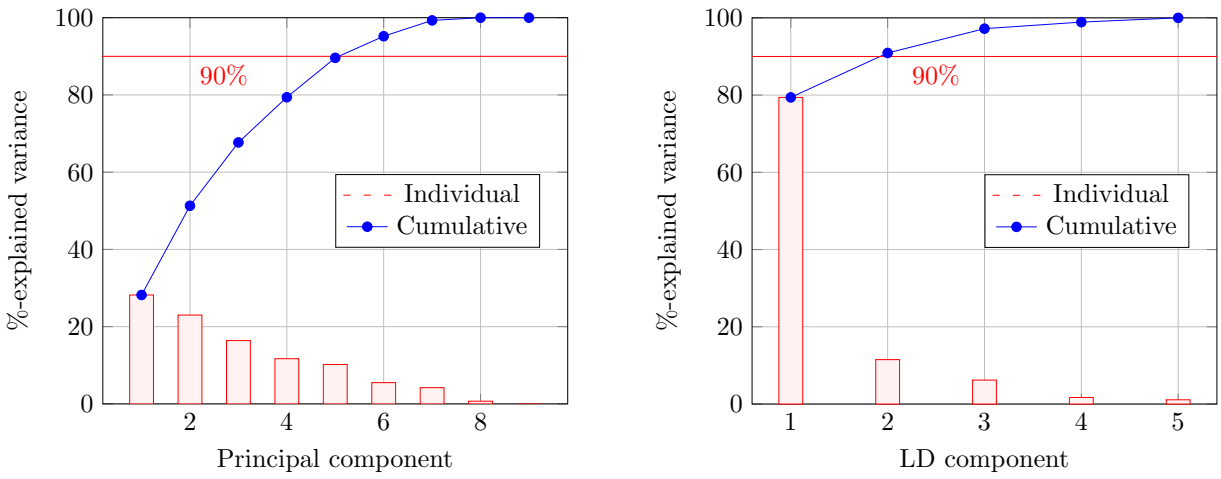

Figure 1: Scree plots of explained variance

Figure 1 shows with two scree plots to which extent each of the new components, which are linear combinations of the original features of the (standardized) train data, capture the variance in the data as the ratio of total variance. We see that the first five principal components combined explain around 90% of the variance (89.6%). Similarly, the first two LD components explain 90.9% of the variance. It therefore seems feasible that we can proceed to classify with fewer features without losing too much information. When we do train models with a subset of features, we may risk losing some performance in cross-validation, but the upside is a smaller risk of over-fitting the model to the train data. In the following, we will look at the visualizations with each method.

2.1 Dimensionality Reduction with LDA

LDA takes the features of the standardized train data to produce linear combinations that maximises between-class variability and minimizes within-class variance. The limited scope of this paper does not allow for a thorough walk-through of the mathematics behind the method, but it is, roughly, a matter of calculating the means of each feature per class and on the basis of these calculating the within- and between class scatter matrices and then solving the eigenvalue problem of the product of the transposed within-class scatter matrix and the between class scatter matrix. We then rank the eigenvectors in descending order by their corresponding eigenvalue (figure 1, right) and construct a coefficient matrix of the eigenvectors that we can use to transform our original features into LD features.

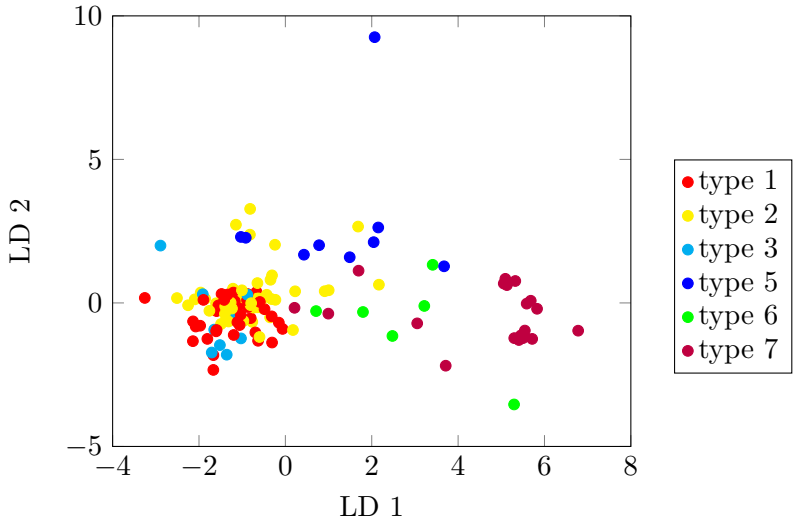

Figure 2: Scatter plot of LD components 1-2

Figure 2 plots the train data points against LD components 1 and 2, and there are a few take-outs from this representation. Window-glass, i.e. types 1, 2 and 3 seem to cluster together in the left side of the plot, types 5 and 6 somewhat in the middle, although not as muddled together as the former 3 types, and the majority of type 7 glass data points seem to cluster in the lower right side of the plot.

These observations indicate that our models - particularly k-NN - will be able to predict type 7 glass as well as window type glass vs non-window type glass as groups given these two LD components. Hence, this is the main rationale for including this dimensionality reduction in our analysis.

2.2 Dimensionality Reduction with PCA

In contrast to LDA, PCA produces linear combinations that maximises variability and is unsupervised. Hence, data may be more spread out in one direction but clustered in another. We see this dynamic in figure 3.

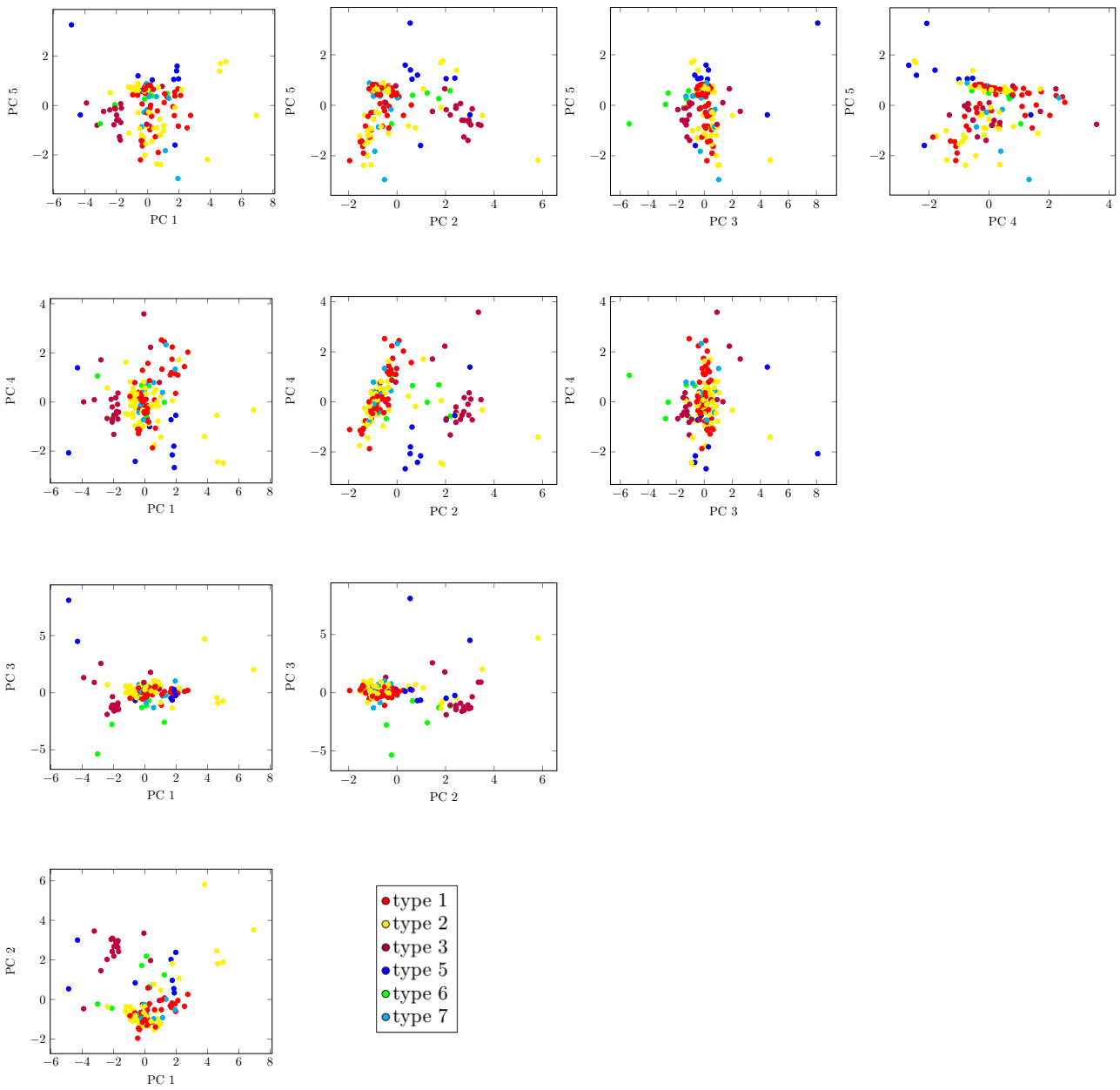

Figure 3: Directions of high variability

The subplots in figure 3 show the first five principal components plotted pairwise against each other. Similarly to LDA, we see a tendency of window-type glass clustering together, but the variance of the data points of a certain class differs between the individual projections - as an example consider PC's 1 and 2 which capture most of the variance of the data set. Neither of these projections manage to spread out type 1 data points relative to other types as well as PC 4, hence we include five principal components for the PCA based models.

3 Classification

3.1 M1 - QDA Implementation

Quadratic **D**iscriminant **A**nalysis (QDA) is a method that can be used for classification by predicting the class that maximises the quadratic discriminant function for each observation shown in equation (1). The QDA method is closely related to LDA, they share the assumption that features of the data are drawn from a univariate Gaussian distribution and that classes have individual means. However, where the LDA assumes that the classes share a common covariance matrix, QDA relaxes that and allows for an individual covariance matrix for each class. Since the class covariance matrices in the given data differ by a substantial margin, we've chosen to implement the QDA method for classification.

The method is carried out by first finding the class specific parameters; class means, μ_k, the class covariance matrices, Σ_k, and the class prior probabilities, π_k, where $k = \{1, ..., k\}$ and k is the number of classes. These parameters can then be used to calculate the discriminants of the classes for each observation given by this quadratic function:

$$g_k(x) = a_k + b_k^T x + x^T c_k x \tag{1}$$

where the coefficients a, b, c are:

$$\begin{aligned} a_k &= 2 \log \pi_k - \log |\Sigma_k| - \mu_k^T \Sigma_k^{-1} \mu_k + const. \\ b_k &= 2\mu_k^T \Sigma_k^{-1} \\ c_k &= -\Sigma_k^{-1} \end{aligned}$$

By assuming individual class covariance matrices instead of a shared one, the discriminant becomes a quadratic function. This also makes the decision boundaries quadratic, which for class j and k for some observation x is given by:

$$g_j(x) = g_k(x)$$

where each side of the boundary is the area of one of the classes, in which an observation will classified as that particular class [2].

For this method we've decided to transform our data to the five first components of our PCA method. This is done to maximize variance and eliminate collinearity in our variables for optimal classification performance from the QDA. Before doing the PCA, we standardized the data by subtracting the mean of each feature from the observations and then brought it to unit variance by dividing the observations by the standard deviation of each feature column. This was done to assert that all features would be equally weighted in the PCA. These preprocessing steps are needed to reproduce the results of our approach.

As explained earlier, to do the QDA we compute the class parameters from the transformed data. For each class we take all of its observations from the training data and compute the mean of all

features, the covariance matrix and the class prior probability. Then, we classify each observation from the test data by using the discriminant function shown in (1) by choosing the class that maximizes the function. All of the predictions are then stored to compute a confusion matrix, from which we draw metrics such as accuracy, precision, recall and F1 scores.

Since vanilla quadratic discriminant analysis has a closed-form solution for each observation and no hyperparameters to tune, the results from our implementation are easy to reproduce, when the class specific parameters are found and the discriminant function (1) is used to classify the observations to the classes that maximizes the function.

To ensure the correctness in our own implementation, we have compared our PCA transformation and QDA classifier with the existing implementation of these methods in the Python library, scikit-learn. Thus, we can given our results confirm, that our implementation does in fact give the same results for this specific research. We thus assume our implementation correct.

The QDA method has the advantage of computing quadratic decision boundaries given the discriminant function shown in equation (1). However, this does not necessarily provide a noticable difference in classification performance when two or more classes tend to be similar as seen with the window classes (types 1-3) in figure 3. We expect our QDA method to perform poorly on the window data, since it is hard to separate the classes as their observations look much alike. However, we expect it to perform better on the non-window classes, as they have more individual tendencies in their parameter values and are thus more separated.

From the classification results we can see that some issues arise in the construction of decision boundaries for the classes in the data. The QDA is performing poorly on the window classes (class 1-3), where it tends to classify all windows as one type. This is indicative of the decision boundaries being in favor of one class in a region where the other classes are present as well. However, this behaviour is to be expected, since it is not possible for the method to construct well-performing classification boundaries on classes that are spread in the same region. We also see, that the fragment type 6 is getting misclassified entirely as type 2 and 7, which is indicative of the decision boundaries of that class being pinched narrow by the other classes. This is arguably a product of the type 6 fragments being poorly represented in the data, as well as being spread sparsely into the regions of other classes. Lastly, we certainly see a decent classification on fragment type 7, which is the only class with a high (above 80%) accuracy, precision, recall and thus F1-score. This is indicative of the class being well separated from the other classes, which is ideal to construct decision boundaries that both classify the designated class correctly and avoid wrong classification of other classes.

3.2 M2 - Decision Tree

The decision tree model works by recursively splitting the training data at the feature and the threshold that yield the lowest value of a chosen cost function. This is done by a greedy algorithm that tests

each possible split, and then pushes the split data into left and right child nodes. The splits that yield the lowest value of the cost function at each node define the decision boundaries for the model. When the data at a node cannot be split further, or when it meets a stopping criterion, the particular node becomes a leaf node. When the model is asked to classify a new observation, it pushes the data from the root down through the tree, based on how the observation responds to the split condition at each node, until it reaches a leaf node that classifies the data. The classification is the class with most occurrences in the leaf node, and the class probability is its fraction of all the observations in the leaf node.

The implementation is represented by a binary tree data structure as described in [3]. When the model is trained, the root node of the binary tree is filled with all the training data, which is then recursively split into two subsets, one for each child node. The two-way split is determined by the CART cost function and the Gini impurity as described below.

When the model is introduced to new data, it uses Gini impurity as a criterion to determine which feature to split the data at and at which threshold to split the feature. In practice, the implementation works by testing each $N-1$ split for each feature and then compute the Gini impurity for the left and the right part of the split respectively. The formula for Gini impurity for the ith node (in our case, left or right node) is given by:

$$G_i = \sum_{k=1}^{K} p_{i,k}(1 - p_{i,k}) \tag{2}$$

where K is the numbers of classes and $p_{i,k}$ is the proportion of data points of class k in the ith node. For each possible split, the Gini impurity of the left and right child nodes are combined to compute the CART cost function for that particular split given by:

$$J(k, t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right} \tag{3}$$

where $G_{left/right}$ is the Gini impurity of the left/right subset of the data, $m_{left/right}$ is the number of instances in the left/right subset and m is the total number of data points in the node before the split. In summary, the CART cost function finds the split in the input data that yields the lowest class impurity. The implementation is tested by comparing results with that of SKlearn's decision tree classifier, and by checking samples of the Gini impurity outputs from the functions to see, if they were calculated correctly. The results were similar to those of SKLearn, and samples of the Gini impurity were computed correctly according to equation 2, so we assume that our implementation is correct.

For multiclass classification the model was initialized with a maximum depth of 4 and a mininimum sample split of 5. Then, the model was trained on the given training data and finally evaluated on the given test data. Likewise for binary classification, the model was initialized with a maximum

depth of 4 and a mininimum sample split of 5. In this case, the classes in the training and test data were mapped to two classes (window and not window), and then, the model was trained on the training data and tested on the test data.

Search for hyperparameters: 5-fold cross-validation

The hyperparameters were found using 5-fold cross-validation. Doing so, the training data of n samples was repeatedly split into two; a train set consisting of $n - 5$ samples and a validation set consisting of 5 samples. Then, we iteratively trained models with different combination of hyperparameters on each of the train sets and computed the mean of their accuracies on the validation sets. This was done to ensure that the measurement of different hyperparameter values suffered less from imbalances in the data set. In contrary, a single train-validation split could have created a bias towards certain hyperparameters and therefore might not have generalized well.

Given that the decision tree model is a non-parametric algorithm that does not make any assumptions about the data, a general issue is that it fits the training data very closely, which can lead to overfitting [4] p. 224. To solve this issue, *pruning* is used, which is a common term for methods that limit the growth of a tree, either before or after the tree is created (pre- and post-pruning, respectively). In our case, the decision tree model implements pre-pruning in which certain hyperparameters limit overfitting. The specific hyperparameters chosen are minimum sample splits (which controls the minimum instances of data points in a node before a split) and maximum tree depth. If either of these conditions are met at a given node, then the node becomes a leaf node that predicts the class with the most occurrences. In other words, the hyperparameters control when a given node has to stop splitting the data further, and therefore, when the model has to stop fitting the training data. The tested hyperparameter scales are shown in the table below.

Cross-Validation of Hyperparameters		
Hyperparameter	Minimum Sample Split	Maximum Tree Depth
	1 to 9	1 to 9
Multiclass Winners	4	5
Binary Winners	4	5

Table 1: An overview of the sets of hyperparameter values tested with 5-fold cross-validation, and the winning combinations of multiclass and binary classification, respectively.

No matter how the decision boundaries must be formed to split classes in a given data set, the decision tree model will do so by simply finding more and more split conditions in the data. In

other words: Any decision boundary shape can be formed by infinitely small split conditions (only limited by the number of training observations). This is an advantage of the model, which may help explain why it performs better than any of the other models on multiclassification (except voting classifier). The model can create narrow decision boundaries! However, it also means that without pruning, the model will have a low bias and overfit the training data by creating decision boundaries between classes, even when it may not be logical. Given that our visualizations show, that some classes may be easily separable on some inputs, the decision tree model ought to perform well here without overfitting. On the other hand, some of the classes are clustered together in the input space, even when the dimensions are reduced by PCA or LDA, and therefore we expect the decision tree model to perform best when overfitting is limited by some hyperparameters.

3.3 M3 - Support Vector Machines

Support Vector Machines (SVM) is a model that classifies an observation by looking at the output of a decision function. In a situation where the classes are linearly separable, the decision boundary will be a straight line defined by the decision function that classifies class 0 or class 1:

$$\hat{y} = \begin{cases} 0, & \boldsymbol{w}^T \cdot \boldsymbol{x} + b < 0 \ . \\ 1, & \boldsymbol{w}^T \cdot \boldsymbol{x} + b \geq 0. \end{cases} \tag{4}$$

where $\boldsymbol{w}$ is the weight vector (with a weight for each feature) and b is the bias.

A margin around the decision boundary is introduced that ranges from $\hat{y} = -1$ to $\hat{y} = 1$. The width of the margin can be controlled by $\boldsymbol{w}$ in the decision function, and the idea is to maximize the margin. If we impose that no observations can be within the margin or on the wrong side of the line (called hard margin classification), then the model may not generalize well because single outliers can lead to a very tiny margin. Therefore our model implements soft margin classification in which observations in the margin or even misclassifications are allowed but minimized. The goal is to find a balance that maximizes the margin while minimizing the margin violations. This is a convex optimization problem solved through constrained optimization, e.g. by using Lagrange multipliers.

In the case of non linearly separable data, kernels that take the data into higher dimensions (without *actually* doing so) are introduced. The "kernel trick" allows the SVM model to stay in the original dimension space by only calculating the dot product of the transformation of the samples, making it computationally efficient. Minimizing this dot product maximizes the margin.

The SVM model was applied using SKlearn's SVC (Support vector classifier) method. Firstly, the data was scaled using Sklearn's StandardScaler which standardizes the features by removing the mean and scaling to unit variance. This was done because the SVM model is sensitive to the feature scales. This makes intuitively sense, when you think about it: The width of the margin should not

depend on the direction of the decision boundary!

For multiclass classification, the model was trained on the given training data with a RBF Gaussian kernel, gamma of 0.1 and C-value of 4/3. The same parameters were used for binary classification except for the C-value which was set to 2/3. Lastly, the two models were tested and evaluated on the given test data.

The hyperparameters were found using 5-fold validation similar to that of the decision tree model to achieve generalized performance measurements. The hyperparameters tested in the SVM model were the kernel, the degree (of the polynomial kernel), gamma (of the kernels) and C-value (which controls to which degree margin errors are allowed). The hyperparameters were tested in the scales seen in the table below.

Cross-Validation of Hyperparameters				
Hyperparameter	Kernel	Degree	Gamma	C-value
	Linear, Poly, RBF	1 to 5	10, 1, ..., 0.000000001	0.33, 0.66, ..., 3.66
Multiclass Winners	RBF	n/a	0.1	4/3
Binary Winners	RBF	n/a	0.1	2/3

Table 2: An overview of the sets of hyperparameter values tested with 5-fold cross-validation, and the winning combinations of multiclass and binary classification, respectively.

The SVM model generally performs well on complex data sets of small or medium size [4] p. 191. Since it is a convex optimization problem, unlike for example Neural Networks, it finds a unique global optimum. The model tends to perform well on data sets, where the number of features do not exceed the number of samples, which is the case for our data set, so we expect the model to perform well.

The fact that the RBF Gaussian kernel in practice yielded a better result than the Linear kernel indicates that the classes are not linearly separable in the input space, and therefore increasing the dimensions improve the classification performance. This means that the decision boundaries are in even higher dimensions than that of the input space, making the boundaries impossible to visualize. This exact versatility of the SVM model, given by the kernel function, may explain why the model performs well on the glass data: The decision boundaries can take many shapes! However, just like for the other models, the clustering of the classes affect the performance. That is why the C-value is useful, because it allows the margin errors, that logically must occur when different classes cluster together, as seen in the PCA and LDA visualizations.

3.4 M4 - k Nearest Neighbours Classification on Two LD Components

k-Nearest neighbours is a classification method with a very simple, general intuition; given a training data set, the model is fitted by placing the observations in an n-dimensional space according to their n features. To classify a new observation, this observation is inserted in the same space and then the labels of the k-nearest neighbors by euclidean distance are observed - the majority label decides the classification and a tie is broken by random [1] p. 126. Each neighbour can furthermore be assigned a weight according to its distance to the new observation - closer neighbours having a stronger vote - or neighbors having uniform weights as is the case in this analysis.

k-NN has been applied to our data first by standardizing it and then transforming it to LD components as described in section 2.1. It is given that we need to choose two features, and we have chosen LD components 1 and 2, as they explain the majority of the variance in the data (figure 1).

For k-NN, the main hyperparameter is k, the number of neighbours from which to cast the vote for labelling a new observation. The chosen k was found by iteratively doing 5-fold cross-validation on standardized, shuffled train data as generally described on p. 7 in the grey box, in this case particularly for k ranging from 1 to 30. This enables computation of average accuracy for each iteration, which along with consideration of the per class precision, recall and F1 scores becomes the basis for the decision of the chosen k, here $k = 4$.

In this particular case, the model classifies data points in the 2-dimensional space visualized in figure 2. Hence, as discussed for the previous models, one can see that it may be hard for the model to distinguish between window-type classes as these are clustered together. On the other hand, as type 7, headlamp glass, in overall is nicely separated from the other classes, one would expect the performance for this particular class to be stronger.

The actual performance of the model when predicting labels of the test data is pretty much aligned with the expectations based on the train data - We reach an overall accuracy of $\sim 60\%$, and the specific metrics are summarized in table 3. We can consider the confusion matrix below:

		True label					
	$\mathcal{C}$	**1**	**2**	**3**	**5**	**6**	**7**
Predicted label	**1**	17	10	2	0	1	0
	2	2	10	3	1	0	0
	3	2	1	0	0	0	0
	5	0	2	0	3	0	1
	6	0	0	0	0	2	1
	7	0	0	0	0	0	7

Columns express true labels where rows are the predictions. Thus, entries on the diagonal are correct predictions. From this, we see that type 2 is often labelled as type 1. As all window-type

observations were clustered together in the train data, this falls in line with the expectations. For the same reasons, the model entirely fails to predict type 3 glass fragments, labelling them as either type 2 or 1. This is further highlighted when considering the decision boundaries for the model in figure 4, where the bounded area for type 3 is indeed very small relative to the other types.

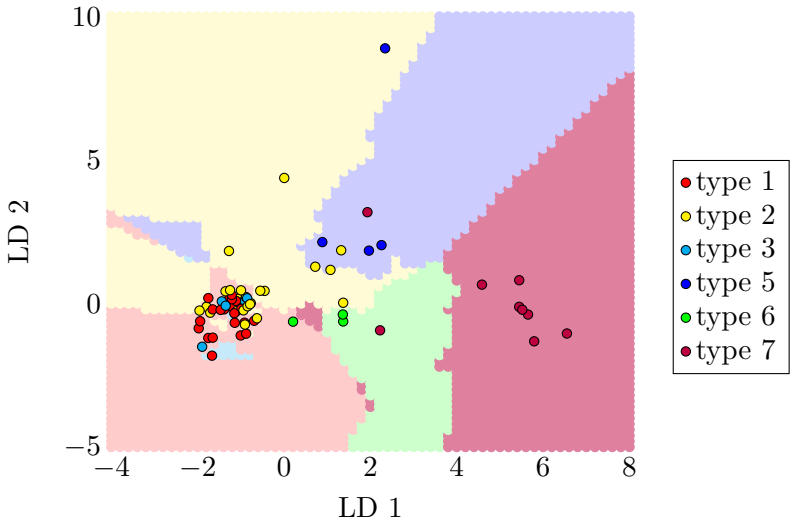

Figure 4: Decision boundaries for 4-NN model

In general, the decision boundaries do not look very well defined, except for type 7 glass fragments - as we initially expected from the visualization in section 2.2. This appears to be a result of the k-NN model relying purely on euclidean distance between data points and these being clustered together for window-type glass.

3.5 M5 - Soft Voting Classifier

The soft voting classifier is an ensemble of models that predicts a class by looking at the highest average probability cast by the committee of models for each observation.

We've applied this method by using the four earlier mentioned models to form the predictors in the ensemble. For the data preprocessing, we have chosen to standardize the data and use PCA to reduce the dimensions of the data to the 5 first principal components, as done in section 3.1 of the report. Again, this was done to eliminate collinearity from the data before applying the QDA method, which is guaranteed by dimensionality reduction with PCA. Lastly, we've chosen the weight of each model by utilizing a brute-force search of all combinations of weights in the range 100-500% for each model. We ended up weighing the QDA, decision tree and K-nearest neighbors classifiers with 300% and the support vector machine classifier with 500%.

The type of hyperparameters were selected for each model in the ensemble by choosing the ones that were used in their individual examination for this project. However, the models were not all trained on the 5 first principal components of the data, thus we've chosen to use grid search to

find the most suiting values. This was done, not by examining the hyperparameters for each model individually, but together in the voting classifier to have the best composite tuning. The grid search was done using a 5-fold cross-validation as described in the grey box on p.7.

Generally, we expect the soft voting classifier to have a better performance than the individual models, since shortcomings specific to a single model will be overruled by the probability votes cast by other models. Also, if the vote of a model is uncertain, then it will be weighed lower, given that each of the votes per model is an estimated probability of the classes. However, it should be mentioned, that if a single model performs exceptionally well on classifying observations that the rest of the models do not, then that strength will be diluted by the combined vote. Ideally, the models will be uncertain, when they classify incorrectly and certain, when they classify correctly. It is also to be expected, that if all models tend to classify the same patterns wrongly, then this will translate to a flaw of the voting classifier, since all of the votes will be affected by this.

The decision boundaries from the soft voting classifier will be a combination of those of the previous models, where regions are defined by the class that has the highest combined probability in the votes cast by the committee.

		True label					
	$\mathcal{C}$	**1**	**2**	**3**	**5**	**6**	**7**
Predicted label	**1**	21	1	3	0	1	0
	2	0	21	2	1	0	0
	3	0	0	0	0	0	0
	5	0	1	0	3	0	1
	6	0	0	0	0	2	0
	7	0	0	0	0	0	8

Through our results (see the confusion matrix above), it can be seen that the probability votes cast by the committee was able to consistently classify the building windows (fragment type 1-2). However, the last window class (fragment type 3) was entirely classified as the just mentioned building windows. This can be assumed to be a product of the models in the ensemble all being more uncertain about this window fragment type than the others, which is consistent with the fact, that the other models also often misclassify this fragment type. This will cause the decision boundaries of this classifier to sway towards the fragment type 1 and 2, and maybe even omit fragment type 3 entirely. The uncertainty is most likely caused by the 3 window fragment types clustering together as seen in the previous figure 2 and 3.

Also, the models have classified the remaining 3 non-window fragment types (5-7) with only very few false negatives. This is indicative of the probabilistic decision regions of the model being very well-defined outside the window fragment cluster. It would seem like the ensemble of the individual models used in this project, apart from fragment type 3, is able to provide reasonable decision boundaries for

all fragment types with only a few observations falling out of their designated class regions.

4 Discussion and Considerations

The final multiclass classification performance for each of the models is presented in the table below:

Multiclass Classification Performance of the Models			
	Accuracy	Macro F1	Weighted F1
QDA	60%	47.9%	55.1%
Decision Tree	80%	74.1%	80.5%
K-Nearest Neighbours	60%	55.4%	58.6%
Support Vector Machines	76.9%	64.5%	74.2%
Ensemble	84.6%	69.9%	81.4%

Table 3: Performance metric comparison between models

As seen in table 3, the decision tree and SVM models have the second and third best accuracy scores, respectively. As discussed in section 3.2 and 3.3, the two models are very flexible in their decision boundaries, and especially the decision tree model can separate classes even when they are very similar - as opposed to k-NN which strictly suffers from being based solely on the euclidean distances to data points with differing labels i.e., in clusters it becomes a "coinflip" situation no matter the k.

The soft voting classifier has an even better accuracy score, which, like we discussed in section 3.5, may be due to the fact, that the model overcomes the uncertainty of the individual models by weighing the combined probability of each model's class prediction. This means that the soft voting classifier has the highest fraction of correct prediction of all the models. In simple terms, the best model predicts the type of glass fragment correctly 84.6% of the time!

Looking at the macro F1 scores, the results are similar but not quite identical. The decision tree model achieves the best macro F1 score, while the soft voting classifier comes second. The macro F1 score shows how well a model performs in a harmonic mean of the precision and the recall for each class, divided by the number of classes. With precision being the fraction of correct predictions, when a model predicts a given class, and recall being the fraction of correct classifications over the number of predictions of a specific class. In summary, the score shows that the decision tree model has the best performance when all classes are treated as equally important.

		True label					
	$\mathcal{C}$	**1**	**2**	**3**	**5**	**6**	**7**
Predicted label	**1**	20	14	4	0	0	0
	2	0	7	0	1	1	0
	3	1	1	1	0	0	0
	5	0	1	0	3	0	1
	6	0	0	0	0	0	0
	7	0	0	0	0	2	8

QDA

		True label					
	$\mathcal{C}$	**1**	**2**	**3**	**5**	**6**	**7**
Predicted label	**1**	18	1	1	0	1	0
	2	1	18	2	0	0	0
	3	2	1	2	0	0	1
	5	2	2	0	4	0	0
	6	1	1	0	0	2	0
	7	0	0	0	0	0	8

Decision Tree

		True label					
	$\mathcal{C}$	**1**	**2**	**3**	**5**	**6**	**7**
Predicted label	**1**	18	1	2	0	1	0
	2	3	19	3	1	0	1
	3	0	0	0	0	0	0
	5	0	2	0	3	0	0
	6	0	1	0	0	2	0
	7	0	0	0	0	0	8

SVM

		True label					
	$\mathcal{C}$	**1**	**2**	**3**	**5**	**6**	**7**
Predicted label	**1**	17	10	2	0	1	0
	2	2	10	3	1	0	0
	3	2	1	0	0	0	0
	5	0	2	0	3	0	1
	6	0	0	0	0	2	1
	7	0	0	0	0	0	7

k-NN

		True label					
	$\mathcal{C}$	**1**	**2**	**3**	**5**	**6**	**7**
Predicted label	**1**	21	1	3	0	1	0
	2	0	21	2	1	0	0
	3	0	0	0	0	0	0
	5	0	1	0	3	0	1
	6	0	0	0	0	2	0
	7	0	0	0	0	0	8

Soft Voting

Figure 6: Confusion matrices from the classifiers in the project

Given that the data set is imbalanced, in that some glass types are more frequent than others, it may make sense to look a weighted F1 score instead. For example, the soft voting classifier and k-NN do not predict type 3 glass at all - as seen in the figure 5 confusion matrices, while the decision tree model does so twice. This influences the macro F1 scores a lot, even though the particular class has a low occurrences in the data set. The weighted F1 score works like macro F1, but instead it weighs each class by its fraction of occurrences in the data set. The soft voting classifier has the highest weighted F1 score, meaning it performs best on the weighted, harmonic average of precision and recall for each class.

The question of which metric to evaluate depends on the forensic purposes. While accuracy generally is a good indicator of model performance (true positives and true negatives), the F1 scores weigh in misclassifications (false positives and false negatives). To sum up, one can ask the questions:

- Are you simply interested in the model, that predicts the highest percentage correctly? Then you may use the accuracy winner (Soft voting classifier)
- Are you interested in the model, that has a high accuracy, but penalized by misclassifications on smaller classes? Then you may use the macro F1 winner (Decision tree)
- Lastly, are you interested in the model that scores the highest on the prior conditions but with

class importance weighed by its fraction of occurrences? Then you may use the weighted F1 winner (Soft voting classifier)

But what if, for forensic purposes, we are primarily interested in discerning whether a glass fragment stems from a certain group, e.g. being a window glass fragment or non-window. In that case, the metrics for all models improve drastically as seen in table 4:

Binary Classification Performance of the Models			
	Accuracy	Macro F1	Weighted F1
QDA	95.4%	93.9%	95.4%
Decision Tree	95.4%	93.9%	95.4%
K-Nearest Neighbours	89.2%	86.3%	89.5%
Support Vector Machines	90.8%	87.6%	90.8%
Soft Voting Classifier	95.4%	93.9%	95.4%

Table 4: Comparing the performance of the models predicting window/non-window types

In this case, the performance of particularly QDA and k-NN improve because it becomes easier for these model to draw clear decision boundaries between what is essentially two well-separated clusters.

As relates to uncertainty, the probability estimates per class for each model can be computed. In an investigation, this would give an idea of how certain it is that a given prediction is correct and by extension the value of the evidence. However, the data for this analysis comes from an experimental setting, yet the intended application is criminal case work. One thus might consider that although we can with some certainty predict the type of a certain glass fragment, it is not necessarily indicative that the glass fragment originates from a given crime scene. Our models may also get confused by glass types not considered in the experiment.

Similarly, one could imagine that the chemical composition of a sample may be influenced by external factors in a real-life setting, which ultimately may lead to misclassification.

References

[1] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] T. Graversen, *ML Lecture notes week 14*. From LearnIT, IT University of Copenhagen, 2020.

[3] R. Sedgewick, *Algorithms*. Addison-Wesley Professional, 2011.

[4] A. Geron, *Mands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media, 2017.