Machine Learning Report

Group38

Matura Tanja (01307001), Morando Fausto (12508079), Drepenteris Stephan (12503833)

November 10, 2025

Contents

1 Introduction

The task of this exercise is to evaluate 3 models on 4 diverse datasets, two of which were provided via Kaggle and 2 that our team selected. The 3 chosen classifiers are:

- **Logistic Regression (ElasticNet):** A linear model that estimates class probabilities using a logistic function. ElasticNet regularization combines L1 (Lasso) and L2 (Ridge) penalties to balance feature selection and stability. It serves as a strong and interpretable baseline.
- **Decision Tree:** A non-parametric model that recursively splits the feature space into homogeneous regions. It captures nonlinear interactions, is interpretable, and robust to mixed data types, though it may overfit small datasets without pruning.
- **Support Vector Machine (SVM):** A kernel-based classifier that finds the optimal separating hyperplane between classes by maximizing the margin. SVMs are particularly effective for small- to medium-sized datasets with complex, potentially non-linear decision boundaries.

This combination allows for evaluation of three distinct learning paradigms: linear, tree-based, and margin-based approaches. More models might be included in the section of a dataset, if they were also tested on the data. An Overview of the results can be found in Section "summary".

To ensure reproducibility random seed was fixed to 42.

2 Dataset 1: Credit Score

The first data set we looked at is the `analcatdata_creditscore` dataset from OpenML[3]. The dataset comes from the book 'Analyzing Categorical Data', by Jeffrey S. Simonoff, Springer-Verlag, New York, 2003[5] and provides data about the Credit Score of 100 people. A more detailed look at the columns is given in the next section.

Our goal is to predict whether someone's credit application will be approved based on information about them. For this we will use logistic regression, a decision tree and a support vector machine. A comparison of results can be found in 2.6 Summary.

2.1 Overview

The dataset is small with only 7 columns and 100 entries. All features are numeric, although `Derogatory.reports` represents an ordinal variable. The target is `Application.accepted`, a binary variable that indicates whether someone's credit application was approved or denied.

This table gives an overview over the features and the target:

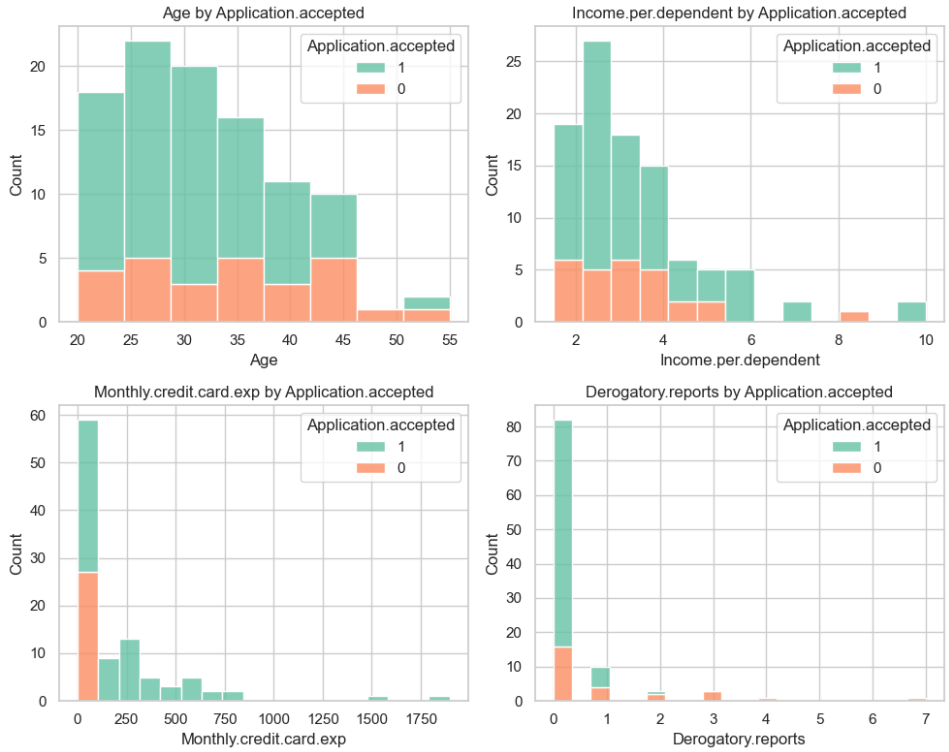
It is notable that **the target variable is skewed**, with 73 being accepted and 27 rejected. This is even more true for `self.employed` (5 yes to 95 no). `Monthly.credit.card.exp` also seems to contain extreme values, with the range being 0-1898 with a mean of only 189. We will take a closer look at this variable later.

`Monthly.credit.card.exp` shows high variance in this snippet and is a candidate for transformation later.

Feature	Types	Observation
`Age`	Numeric (Integer)	Range 20-55, mean 32
`Income.per.dependent`	Numeric (Integer)	
`Monthly.credit.card.exp`	Numeric (Integer)	Range 0-1898, mean 189
`Own.home`	Categorical (Boolean)	64 Yes, 36 No
`Self.employed`	Categorical (Boolean)	5 Yes, 95 No
`Derogatory.reports`	Numeric (Integer)	Range 0-7
Application.accepted	Categorical (Boolean)	73 Yes, 27 No

The categorical columns - that are binary and therefore encoded as `0` and `1` - and `Derogatory.reports` are still byte enconded and had to be decoded first before they could be worked with.

Target based distribution of numeric features:



The bar plots show some extreme values for `Monthly.credit.card.exp` and `Derogatory.reports`. However, neither seem to be unrealistic and therefore not errors in the data set but real extreme values.

Something else that stands out is that both of these features seem to be highly correlated with the target variable - the application of people with high monthly expenses were accepted, while the applications of people with more than two derogatory reports were rejected. To reduce the influence on the prediction we will transform these features before fitting a model (see Pre-Processing).

`Age` and `Income.per.dependent` also show right skewing and might be transformed later.

Target based distribution of binary features:

The plots for the binary features show that the `Self.employed` feature is very unevenly distributed - there is almost no data for people who are self-employed. Their application rate also goes against the trend, with more applications rejected than accepted.

2.2 Data cleaning and pre-processing

After gaining a better understanding of the data we are working with, we prepared the data to be suitable for model fitting.

Byte Decoding: Some columns were still byte-encoded after importing the .arff file and had to be decoded first.

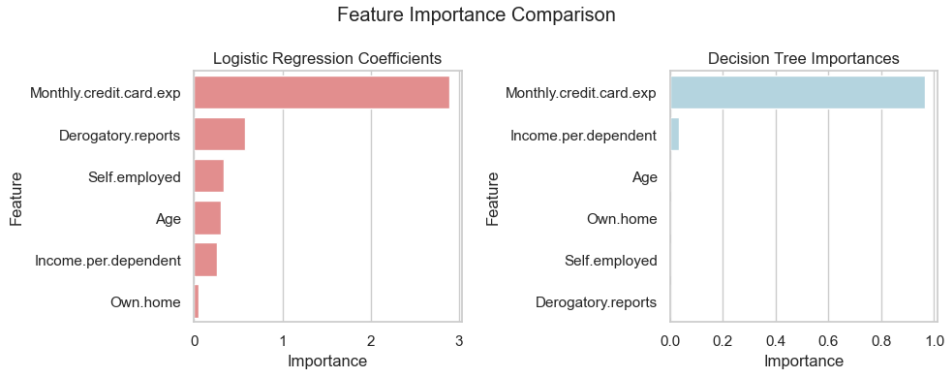
Missing Values: The Credit Score Dataset is complete and therefore didn't need any handling of missing values.

Normalization of Binary Features: To make sure no entry is missed during conversion (in case some binary features aren't encoded as 0 or 1) we searched the columns for subsets of `Yes`, `No`, `True` and `False` and converted them to 0 and 1, should any be found.

Conversion of Numeric Values: All numeric values in the dataset were converted from String to ensure later calculations work. For this the pandas function `pd.to_numeric` was used, which detects and converts numeric Strings automatically.

Outlier Handling: Some rows contain extreme values. At first we considered to keep them, however after beginning to work with the dataset an issue arose: Even after log transforming `Monthly.credit.card.exp` correlated so strongly with the target that it essentially encoded the same information - leading to unrealistic 0% prediction error in the first attempts using Logistic Regression.

Visual diagnostics confirmed that this feature is dominating the others.

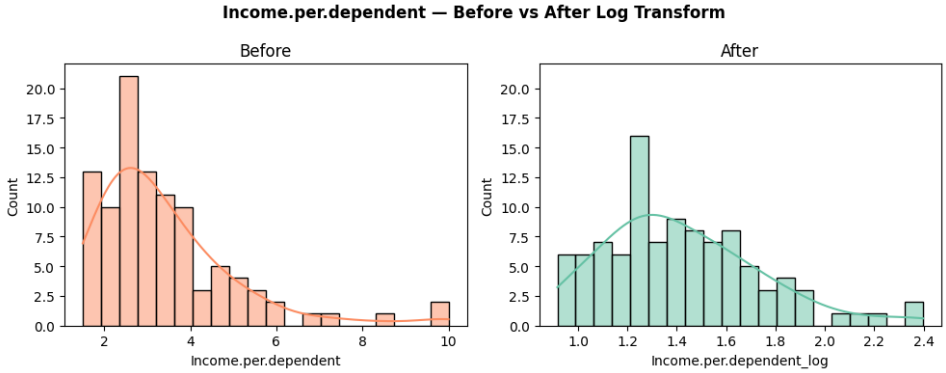


To reduce the influence of `Monthly.credit.card.exp` we decided to cap the column. For this we used the IQR: The upper treshold turned out to be 632.01 and 6 rows needed to be capped at this value. After capping the extreme values in `Monthly.credit.card.exp`, the performance of the Logistic Regression model decreased from 100% accuracy to a more credible level of 90-95%.

Transformation of Numeric Values: Instead of relying on visuals only we used the function `skew()` to output the skewness of each numeric feature. Features with $|\text{skew}| > 1$ were considered strongly skewed. This applied to `Monthly.credit.card.exp` and to `Income.per.dependent`.

After log transformation the features changed as followed:

Feature	Skewdness before	Skewdness after
`Income.per.dependent`	1.882	0.800
`Monthly.credit.card.exp`	1.220	-0.586



Scaling of Numeric Values: Numeric features were scaled using RobustScaler to mitigate the influence of remaining outliers.

Class Balance of Binary Features: Our binary columns (including the target) show class imbalance. Features were kept as they are, the imbalance of the target will be adressed by using `class_weight='balanced'` during model fitting.

Training- and Test-Split: 80:20

2.3 Model 1: Logistic Regression

As baseline model, we first trained a Logistic Regression classifier to predict `Application.accepted`. Logistic Regression is well-suited for this dataset because the target variable, `Application.accepted`, is binary (approved vs. rejected), which fits the model's purpose of predicting probabilities for categorical outcomes.

Since the target variable was moderately imbalanced (approximately 1:3 ratio), two balancing strategies were compared: First, assigning class weights inversely proportional to class frequency (`class_weight='balanced'`), and second, random oversampling of the minority class to equalize class counts.

As discussed earlier the model achieved 100% accuracy on the training set before `Monthly.credit.card.exp` was capped. After capping and re-fitting the model, performance metrics became more realistic.

Both balancing methods produced identical results, with the model achieving an overall accuracy of 85%. The recall for rejected applications (Class 0) remained perfect, indicating no false negatives, while the recall for accepted applications (Class 1) slightly decreased to 0.8, suggesting more realistic model behavior. The model now generalizes better and provides a more balanced interpretation of multiple predictors rather than relying on a single dominating feature.

2.4 Model 2: Decision Tree

The second model we trained was a decision tree. Since the dataset is relatively small (only 100 samples), trees can easily overfit if left unpruned. Therefore, we limited the depth and minimum number of samples per leaf to help ensure better generalization.

Even with this the tree achieved a perfect prediction score on it's first run - but later cross validation showed that the model does not always predict with perfect accuracy, suggesting that the training and test data split happend to make the dataset perfectly separable. This is again due to the variable `Monthly.credit.card.exp`. All applications that were rejected did not have any expenses, making it a very strong predictor for a tree (if expenses $0 \Rightarrow$ reject). However, some people without expenses still received a positive application result, which means the feature cannot be deterministic factor and a realistic model should not achieve 100% accuracy.

2.5 Model 3: Support Vector Machine

The third model that was trained for the Credit Score Dataset was a support vector machine. Instead of just any line that separates accepted and rejected applications, it chooses the one that maximizes the margin — the distance between the boundary and the nearest data points from each class (called support vectors). This makes it robust to noise and reduces overfitting, especially in small datasets. Its margin-based approach makes it more generalizable than a Decision Tree and more flexible than Logistic Regression when the separation between classes is not perfectly linear.

The SVM performed well overall but didn't reach the near-perfect performance of the Decision Tree.

With class weights, accuracy was 80%, and the minority class (rejected applications) had a lower precision (0.56), indicating some misclassifications.

With oversampling, performance improved to 90% accuracy and a macro-F1 of 0.88, showing better class balance.

2.6 Summary

Since we established that the target variable is imbalanced we chose Stratified K-Fold cross-validation. Working with a small dataset also meant that we should keep k small, so the validation sets won't become too small.

Model	Accuracy (CV)	macro-F1 (CV)
Logistic Regression	0.970 ± 0.024	0.964 ± 0.030
Decision Tree	0.980 ± 0.024	0.976 ± 0.029
SVM	0.880 ± 0.050	0.860 ± 0.060

Table 1: Performance of the Credit Score dataset (CV results).

While the Decision Tree achieved the highest apparent accuracy, this likely reflects overfitting. Logistic Regression achieved similarly strong results but with better generalization. The SVM showed lower scores yet provided the most balanced trade-off between flexibility and robustness.

3 Dataset 2: Vehicle

The second dataset is provided by OpenML and simply called `vehicle`[4]. The data describes the silhouettes of vehicles and the purpose is to classify a given silhouette as one of four types of vehicle (`opel,saab,bus,van`), using a set of features extracted from the silhouette.

3.1 Overview

The dataset comprises **846 instances**, each with **18 numerical features**, many of which are significantly **correlated**. Preliminary inspection of the feature names indicate that these features likely represent higher-order shape descriptors (e.g., volumetric or moment-based measures) derived from vehicle silhouettes.

Given the nature of the features, some of them may be redundant or **contribute marginal gains** relative to their computational cost. A correlation matrix analysis revealed multiple attribute pairs with correlation magnitudes greater than 0.7, though never 1. This suggests that are strongly related, but not linearly dependent. It is further assumed that the correlated features encode some kind of **direction-depended tensor information**, that results in substantial, but not unary correlation magnitudes.

To address the classification task and due to the high-dimensional, multiple instance dataset, the models considered were: Support Vector Machine (SVM), Random Forests (included for documentation purposes only), K-Nearest Neighbours (KNN) and Logistic Regression.

These models offer insights to effectiveness, interpretability and computational cost tradeoffs.

3.2 Pre-processing

In the context of finding the best fitted model the metric for performance used was the 10 fold cross-validation (CV) score.

CV Variability: Preliminary tests showed high variability of CV scores (verified also by other performance metrics) depending on the selected training/test dataset split. This sensitivity was addressed by testing multiple of the most common (2/3-1/3, 80/20, 90/10 as well as intermediate ratios), using the **train_test_split** routine, reproducible with a fixed random seed 42. The **stratify** feature was also used, to maintain the statistical characteristics of the original dataset when creating subsets. This ensures that the performance metrics produced for the best models are comparable with each other.

After selecting a candidate model for training, in order to identify the optimal setting, multiple values for all relevant hyperparameters were inserted into a **parameter_grid**, as reference for a pipeline of training. Then using the routine **GridSearchCV** the best fitted model for each dataset split, as well as the hyperparameters used where identified. The performance of each candidate best model was verified by testing on the same, held-out set of data using standard performance metrics: Accuracy, Precision, Recall and F1 score.

Train/Test split: For the majority of models, when trained on more that ca. 700+ instances showed diminished effectiveness, suggesting overfitting or bias introduced by the imbalanced tail-end of the otherwise balanced dataset. This observation is further related to the following remark:

Fit/Validation Tradeoff: CV score becomes less accurate as the validation size decreases. For splits with >90% of the data used for training, CV often overestimates model fit. This is a result of an unavoidable tradeoff, between training depth and validation integrity.

Performance Visualization: As discussed, although the localized CV is in certain cases an unreliable indicator of actual performance, a plot of the CV vs the dataset percentage used for training produces an insightful visualization of dataset regions with particular characteristics, such as localized noise that may or may not help the model by increasing its robustness.

Principal Component Analysis (PCA): In an effort to reduce the computational effort as well as potential outlier values that were difficult to otherwise identify due to the attribute vagueness, the PCA technique was used to combine highly correlated features and reduce the dataset dimensionality, maintaining in a good degree the dataset variance. Multiple variance thresholds resulting to different number of components were selected for each run. The performance scores where in most cases improved by approximately 5%. The unavoidable cost is the complete loss of interpretability, as the components used as input for the model have no actual meaning or relation to the original 18 attributes.

Feature Selection with Recursive Feature Elimination (RFE): This method was inspired by the Feature Redundancy Hypothesis discussed previously, with a goal of ranking the features by importance and incrementally including them to assess their individual contribution to model performance. The iterative inclusion strategy allowed for controlled evaluation of each feature's utility, balancing predictive gain and additional computational load.

This approach did not lead to noticeable improvements, by discarding the least contributing feature for the particular model 'Kurtosis about major' and reducing the dataset by one dimension yielded at best `[Accuracy=0.78, Precission=0.78, Recall=0.78]` for the noticeably different setting `[entropy, max depth=10, max features = log2, min samples leaf=1, mi samples split=2, n estimators=100]`

3.3 Model 1: Logistic Regression

Logistic Regression was selected as a baseline model for the classification task due to its efficiency and robustness under highly correlated features providing also linear decision boundaries.

For the hyperparameters, three types of **penalties** (L_1, L_2, elasticnet hybrid) and a total of five magnitude scales for the regularization parameter `C` (1e-2, ... , 1e+2) were tested. Further tests showed that there was no usable model with regularization $>$ 1e+2. In order to not introduce additional computational load, a **solver** (saga) compatible with both L_1, L_2 norms was chosen and finally only for the elasticnet penalty, a **ratio** (0.15, 0.5, 0.85) of the two norms had to be included in the **parameter_grid**. To ensure convergence, a very high max_iter=1e4 was also set.

A surprising observation was that the hybrid model did not dominate in the best candidate models per CV, with all penalties appearing equally often, but L_2 showed more consistent results with different regularization parameters. The best predicted CV model was `[C=100, p=L2]` trained on 95%, once again displaying a high training size bias and the actual recorded best was `[C=10, ratio=0.5]` trained on 70% of the dataset and yielding in the following performance metrics `[Accuracy=0.85, Precision=0.85, Recall=0.85]`

Although the actual size of the test data varies between splits, using stratify we ensure that each test set has similar statistical characteristics and belongs to a fixed test superset. Assuming it is sufficiently populated (as in this case with no deficiency warnings) we can assume the scores are comparable, keeping in mind that smaller test sets may introduce slightly larger standard deviation.

3.4 Model 2: Decision Tree Classifier

To complement the previous model, a Decision Tree Classifier was trained and evaluated on the vehicle dataset. The model was tuned using a grid search across several key hyperparameters: `criterion` (`gini`, `entropy`), `max_depth` (None, 10, 20, 30, 40), `min_samples_split` (2, 5, 10), `min_samples_leaf` (1, 2, 4), and `splitter` (`best`, `random`). Cross-validation was conducted with 10 folds over various training proportions (from 65% to 95%), mirroring the procedure applied to the other classifiers.

The best-performing configuration was found at a training proportion of 95%, with the following parameters: `criterion=entropy`, `max_depth=10`, `min_samples_split=5`, `min_samples_leaf=2`, and `splitter=best`. Under these settings, the Decision Tree achieved an overall accuracy of **70%** on the validation set, with a macro-averaged F1-score of **0.71**.

A detailed breakdown of the per-class performance is shown in Table 2. The model performed particularly well on the *bus* class (F1 = 0.91), while its performance was more modest on the *opel* (F1 = 0.48) and *saab* (F1 = 0.56) classes, indicating some confusion among visually similar vehicle types.

Table 2: Decision Tree classification report for the vehicle dataset.

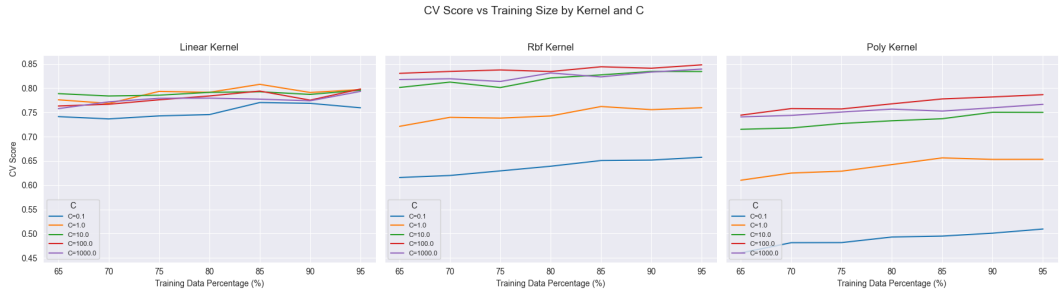
Class	Precision	Recall	F1-score	Support
Bus	0.91	0.91	0.91	11
Opel	0.50	0.45	0.48	11
Saab	0.50	0.64	0.56	11
Van	1.00	0.80	0.89	10
Accuracy		0.70		
Macro avg	0.73	0.70	0.71	43
Weighted avg	0.72	0.70	0.70	43

Overall, the Decision Tree demonstrates a balanced performance with strong interpretability. While it does not surpass the ensemble models in predictive power, its transparent decision boundaries make it a valuable baseline for understanding the structure of the data and feature relevance.

3.5 Model 3: SVM

Support Vector Machines are well-equipped to manage high-dimensional datasets due to their ability to construct optimal decision boundaries in complex feature spaces. Despite their known sensitivity to highly correlated features, a preliminary evaluation was conducted without accounting for the latter, serving as a baseline for effectiveness. Surprisingly, the SVM models yielded strong results, indicating that either the kernel function compensated for the redundancy or that the correlated features retained complementary discriminative value.

For the hyperparameters, three types of **kernels** (linear, RBF, polynomial of degree 2,3,4) and a total of five magnitude scales for the regularization parameter **C** (1e-1, ... , 1e+3) were used. Further tests showed that there was no usable model with regularization $>$ 1e+3 and also the parameter **gamma**, the influence of a single training example, yielded always best results when set to 'scale'. The linear kernel consistently delivered stable and satisfactory performance across multiple training splits. However, certain configurations of the RBF kernel achieved superior results, likely due to their ability to capture nonlinear relationships within the feature space. In contrast, the polynomial kernel consistently underperformed.



It is worth mentioning that the best predicted model per CV score was `[rbf, C=100, gamma=scale]` trained on 95% of the dataset. This is actually an instance where the CV accuracy problem is apparent. By searching all the test runs, the best model had remarkably the exact same hyperparameter setting, but it was trained on 75% of the dataset instead. The produced performance metrics `[Accuracy=0.87, Precission=0.87, Recall=0.87]` are therefore more reliable.

Training with 75% of data. Selected PCA n_components: `'pca_n_components': 0.999, 'svc_C': 100, 'svc_gamma': 'scale', 'svc_kernel': 'rbf'`

Class	Precision	Recall	F1-score	Support
bus	1.00	1.00	1.00	55
opel	0.77	0.70	0.73	53
saab	0.75	0.81	0.78	54
van	0.96	0.96	0.96	50
accuracy			0.87	212
macro avg	0.87	0.87	0.87	212
weighted avg	0.87	0.87	0.87	212

Table 3: Classification report for the SVM model on the Vehicle dataset.

To address the feature correlation problem, the PCA method was used in the same pipeline setting for linear and rbf kernel models, with the performance metrics surprisingly being the same and also resulting for the same dataset size. Therefore PCA offered no advantage.

Comparing the two best performing models (SVM, Logistic regression), by absolute effectiveness the SVM is superior, yet the complexity, as well as the loss of interpretability after the necessary PCA transformation justify the selection of Logistic Regression as the Best Model overall.

3.6 Other Models

3.6.1 Random Forest

Random Forests were considered due to their robustness in handling high-dimensional data and their ability to model complex, nonlinear relationships while mitigating overfitting through ensemble averaging.

Although multiple hyperparameter settings were tested [number of estimators, max depth, min samples split, min samples leaf, max features(sqrt, log2) and criterion(gini, entropy)], the computational load was too large for more exhaustive testing and the sampled effectiveness was underwhelming as well. The best found setup was `[gini, max depth=None, max features = sqrt, min samples leaf=1, mi samples split=5, n estimators=300]` which delivered `[Accuracy=0.76, Precission=0.76, Recall=0.76]`

Class	Precision	Recall	F1-score	Support
bus	0.90	0.98	0.94	65
opel	0.64	0.45	0.53	64
saab	0.56	0.65	0.60	65
van	0.92	0.97	0.94	60
accuracy			0.76	254
macro avg	0.76	0.76	0.75	254
weighted avg	0.75	0.76	0.75	254

Table 4: Classification report for the Random Forest model on the Vehicle dataset.

3.6.2 KNN

K-Nearest Neighbors model was selected due to its effectiveness in capturing local structure without requiring extensive training or separation between features.

For the hyperparameters, apart from the most important number of neighbors, different **weights** (uniform, distance), and L_p **norm** (p=1,2) were used. Further tests showed that there was no usable model with neighbors > 9. The best models had different setups

depending on the split size, but more predominant was the number of neighbors = (4,5,6), the L_1norm and distance weights. Observing the plots, using the L_1norm in most cases yields better results but there is no clear better weight value; although models with distance weights seem to be more consistent I terms of CV score across runs.

Once more, the end-point CV inaccuracy is apparent with the best predicted model per CV score being `[n neighbors=5, p=1, weights=distance]` trained on 95% of the dataset. By searching all the test runs, the best model had actually different hyperparameter setting `[n neighbors=3, p=2, weights=uniform]` and was trained on 85% of the dataset, resulting in `[Accuracy=0.73, Precision=0.74, Recall=0.73]`. These are quite satisfactory metrics considering the simplicity of the model (ca. 3 times more accurate than random guessing).

Class	Precision	Recall	F1-score	Support
bus	0.97	0.97	0.97	33
opel	0.52	0.38	0.44	32
saab	0.53	0.72	0.61	32
van	0.93	0.87	0.90	30
accuracy			0.73	127
macro avg	0.74	0.73	0.73	127
weighted avg	0.74	0.73	0.73	127

Table 5: Classification report for the KNN model on the Vehicle dataset.

In an effort to compress the sample space by exploiting the high cross-feature correlation, once again the PCA method was used in the same pipeline setting. In this case the CV prediction for the best model `[n neighbors=6, p=1, weights=distance, Var=0.998]` trained on 90% of the dataset was closer to the actual recorded best `[n neighbors=7, p=1, weights=distance, Var=0.997]` trained on 85%, which resulted in `[Accuracy=0.78, Precision=0.79, Recall=0.78]`, demonstrating a 5% improvement in all metrics.

The fitness of L_1 as norm is more evident after the PCA application as well as the tendency of the model being more abstract due to the larger overall k (= n neighbors).

3.7 Summary

Model	Accuracy (CV)	Precision / Recall (approx.)
Logistic Regression	0.85	0.85 / 0.85
SVM (RBF kernel)	0.87	0.87 / 0.87
Random Forest	0.76	0.76 / 0.76
KNN	0.78	0.79 / 0.78

Table 6: Performance summary for the Vehicle dataset (best SVM with RBF kernel).

While the process is computationally intensive, it remains non-exhaustive, leaving room for untested hyperparameter and data split combinations that may yield superior model performance. Nonetheless, the use of multiple train/test size configurations enables the estimation of statistical upper and lower bounds for model effectiveness.

4 Dataset 3: Congressional Voting

This dataset was provided via Kaggle[1] and contains data about votes that took place in the US congress.

4.1 Overview

The training set is composed of 218 observations and 17 categorical variables. Each variable refers to a specific topic discussed during congressional voting, and it takes the value *y* if the representative voted in favour and *n* otherwise.

Our goal is to predict the variable `class`, which takes two values: `republican` and `democrat`, using an appropriate classification model. The model is trained on the training dataset, compared with other classifiers, and then evaluated using performance measures. To select the best model, we applied a 10-fold cross-validation and chose the classifier with the highest accuracy, while also considering its interpretability. Finally, we used the selected model to predict the response variable on the test set and submitted the results to the Kaggle platform.

4.2 Pre-processing

Missing Values After loading the training data and looking up to its structure, we notice that several variables contain missing values. Dropping all rows with missing data would reduce the dataset by more than 50%, so we decide to impute the missing values. For such reason, we use logistic regression to predict the missing values of each variable, treating them as unknown outcomes and using all the other variables *(except the real target class)* as predictors. After fitting 16 logistic regression models and imputing the missing values, the training dataset is complete.

To explore the relationship between the response variable and each predictor, we create two-way contingency tables. Below is the table for the variable *physician-fee-freeze*, which shows an almost perfect separation between the two classes.

physician-fee-freeze	democrat	republican
False	122	3
True	5	88

This suggests that this variable will likely play an important role in the following models.

4.3 Model 1: Logistic Regression

The first classifier is a logistic regression model that uses all predictors. By the function `LogisticRegression` from the `sklearn.linear_model` package, we model the probability of voting `republican` given the set of explanatory variables. In the table below, the column `Coefficient`, referring to the full logistic model, shows the direction of the effect (positive or negative) and its magnitude in terms of log-odds ratio. As expected, a positive and huge effect on the probability to vote `republican` is due to the variable `physician-fee-freeze`.

Using the `Kfold` function from the `sklearn.model_selection` package, we obtain a cross-validation accuracy of `0.9677` for the full logistic regression model. Although this is a very good result, using all the features may lead to overfitting and reduces the interpretability of the model.

For this reason, the next step is to force some of the coefficient estimates to be exactly zero by applying an *L1* penalty. In lasso regression, the tuning parameter lambda controls the strength of the penalty: the larger the value of lambda, the stronger the shrinkage

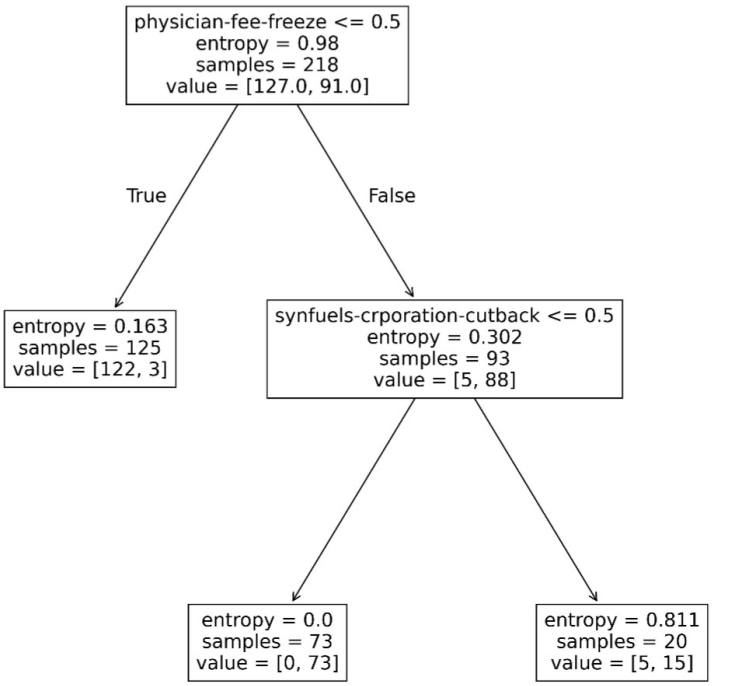
applied to the coefficients. The optimal value of lambda can be selected through 10-fold cross-validation by choosing the value that achieves the lowest prediction error. In our case, we create an equally spaced sequence of 30 values on a log10 scale to search for the best parameter. In our code, C is the inverse of lambda and at the end, we find its optimal value equal to **2.593**. Looking at the column `Coefficien penalized`, we can notice that most features no longer have a direct effect on the response variable, and even the most important predictors show a strong reduction in their coefficient values. The cross validation accuracy for the penalized model is **0.963**. Although this value is slightly lower than the accuracy of the full model, we prefer the penalized version because it is more interpretable and has more degrees of freedom, since fewer coefficients need to be estimated.

No.	Feature	Coefficient	Coef. penalized
0	intercept	3.139287	0.000000
1	physician-fee-freeze	78.985866	7.616169
2	religious-groups-in-schools	-39.341032	-1.039915
3	synfuels-crporation-cutback	-36.084802	-3.548138
4	el-salvador-aid	29.953863	0.000000
5	adoption-of-the-budget-resolution	-23.591818	-2.313113
6	mx-missile	-21.747931	-2.370776
7	water-project-cost-sharing	-20.466584	-1.969317
8	anti-satellite-test-ban	17.734831	0.648073
9	handicapped-infants	13.394403	0.525151
10	education-spending	-11.618886	0.000000
11	export-administration-act-south-africa	7.910343	1.161229
12	aid-to-nicaraguan-contras	-6.469110	0.000000
13	duty-free-exports	6.451421	-1.009904
14	crime	-5.954267	0.113905
15	superfund-right-to-sue	1.790300	0.000000
16	immigration	0.981784	0.167732

Table 7: Comparison of full and penalized logistic regression coefficients, sorted by absolute coefficient magnitude.

4.4 Model 2: Classification Tree

As next classifier, we fit a decision tree by using the function `DecisionTreeClassifier` function, which allows us to specify the splitting criterion. In our case, we use entropy as the node impurity measure. The higher the entropy, the lower the information gained from a given split, so the algorithm prefers splits with the smallest possible entropy. The binary splitting process continues until the stopping rule is reached: a terminal node (leaf) must be pure, meaning it contains only observations belonging to the same class. The resulting tree is very large and consists of nineteen pure terminal nodes, which suggests that the model is overfitting the training data. To reduce overfitting and improve generalization, we prune the tree to obtain a smaller and more interpretable model for predicting the test set. The idea is similar to tuning the penalty parameter in lasso regression. Here, we select the best alpha value *(the cost-complexity pruning parameter)* through cross-validation. The alpha value that maximizes the accuracy among the tested values is **0.0258**. The resulting pruned tree has the first split done by the variable `physician-fee-freeze`, which we have already considered as the most important to predict the vote, and then a less weight is given to the variable of the second split, i.e. `synfuels-crporation-cutback`. At the end, we calculate its 10-fold cross validation accuracy, which is equal to **0.9628.**



4.5 Model 3: SVM

As a third classifier, we trained a Support Vector Machine (SVM) using the `SVC` function from the `sklearn.svm` package. The SVM constructs an optimal separating hyperplane that maximizes the margin between the two classes, making it well suited for binary classification problems such as this one.

Given that the features in this dataset are binary, we used a linear kernel, as non-linear transformations were not necessary.

The model achieved an accuracy of **0.954** and a macro-F1 score of **0.937**, demonstrating excellent predictive performance. These results suggest that the dataset is largely linearly separable — the two political classes can be distinguished by a relatively small number of decisive votes.

Because most variables represent binary "yes" or "no" decisions on specific issues, the SVM could easily identify a clear boundary that separates the voting patterns of Democrats and Republicans. The strong results also confirm that the SVM effectively balances margin maximization and classification accuracy, leading to robust generalization on unseen data.

Compared to the logistic regression and decision tree models, the SVM reached similar performance but with fewer assumptions about the feature relationships, reinforcing its suitability for high-dimensional categorical data with clear class separation.

4.6 Other Models

4.6.1 Naïve Bayes

At this stage, we change the type of algorithm and implement a naïve Bayes classifier. Since our attributes are binary and assumed to be statistically independent and equally important, we use the `BernoulliNB` function to fit the model. After examining the log prior probabilities and the empirical log probabilities of the features given each class, we perform a cross-validation to evaluate the predictive performance. Because of the simplicity of the model and the fact that some of its assumptions are not fully satisfied, the prediction accuracy is the lowest so far: **0.9121**.

4.7 Summary

Model	Accuracy (CV)
Logistic Regression (full)	0.9677
Logistic Regression (L1)	0.9630
Decision Tree (pruned)	0.9628
Naïve Bayes	0.9121

Table 8: Cross-validation accuracy results for the Congressional Voting dataset.

For interpretability and best accuracy, we choose the lasso regression and the pruned tree as best models to predict an unknown test set. After loading it, the next step is to impute its missing value from the models fitted in the training set. Submitting the two predictions in the *Kaggle* competition, the lasso model has a test accuracy approximately of 0.95370, while the pruned tree achieves a slightly better result of **0.96296**.

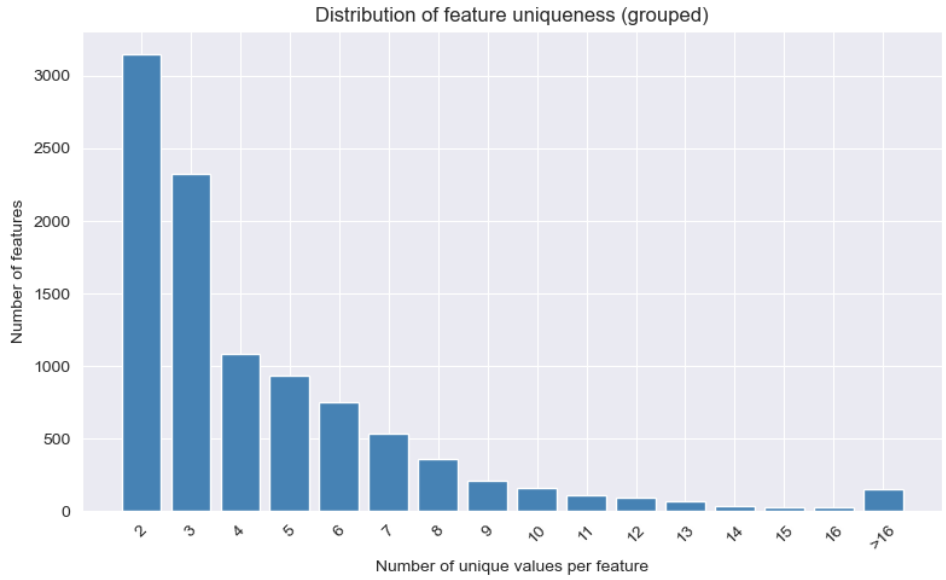
5 Dataset 4: Amazon Reviews

Our fourth dataset also comes from Kaggle and is the `Reviews` dataset [2], which contains data about Amazon reviews and their authors. Because the feature columns are unlabeled, we do not know their exact meaning, but they likely represent linguistic or stylistic characteristics extracted from text (e.g., word frequencies, sentence structure).

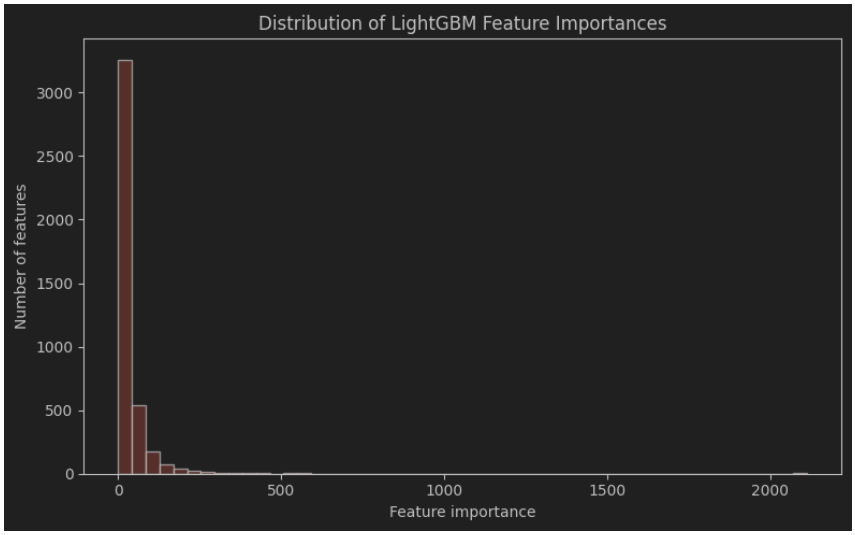
The task of this competition was to predict the `author of a review` based on numerical features derived from review texts. The target variable (`Class`) contained 50 distinct authors.

5.1 Overview

The training data consists of 750 rows and 10,002 feature columns, making it a very wide dataset. All data is already vectorized and might be the output of a preprocessing step that transformed text into numeric representations for machine learning.



The entire dataset contains integers, which might be counts, categories, or ordinal values. This made interpreting the features more difficult as we do not know whether 2 is "twice



as much" as 1. Over 3,000 are binary (two unique values), and more than 2,000 have three unique values. This indicates that the dataset mainly consists of discrete, categorical-like integer features rather than continuous variables.

The target variable (`Class`) spans 50 authors but is quite balanced:

Feature importance analysis using LightGBM revealed that the vast majority of features do not hold predictive power:

In later steps we reduce the number of features both to reduce noise and help with runtime. To understand the nature of the features, feature importances from a linear model (Logistic Regression) were compared with those from LightGBM.

The comparison shows a strong correlation ($r \approx 0.75$), suggesting that most predictive information is linear. However, a subset of features is rated as more important by LightGBM, indicating additional non-linear dependencies and interactions. This means that the dataset contains both linear and non-linear relationships.

5.2 Pre-processing

Label Encoding: Author names were encoded into integer labels using `LabelEncoder`.

Data Integrity: No missing values, duplicates, or constant columns were detected.

Downcasting: All integer columns were downcast from `int64` to `int8`, reducing memory usage from 57.3 MB to 7.2 MB (87.4% reduction).

Feature Categorization: Based on the number of unique values, columns were categorized as *binary* (2 values), *categorical* (2–10 values), or *continuous* (>10 values) for targeted processing.

Dimensionality Reduction: A two-step feature selection pipeline was implemented. First, a Variance Threshold filter (`threshold=0.01`) removed nearly constant features. Next, feature importance ranking using a LightGBM classifier selected the top features explaining 85% of cumulative importance. This preserved most predictive information while greatly reducing dimensionality.

To capture both linear and non-linear relationships we decided to train an ensemble of classifiers combining different linear models, kernel-based methods and tree-based learners. But first we evaluated three core models:

5.3 Model 1: Logistic Regression

Relevant parameters: `penalty="elasticnet"`, `solver="saga"`, `l1_ratio=0.5`, `C=0.5`, `max_iter=1000`.

This model is well suited for datasets where many features contribute weakly to the prediction.

Table 9: Performance of Logistic Regression on full and reduced feature sets.

Dataset	Accuracy	Macro-F1	Runtime [min]
Full	0.513	0.490	397.58
Reduced	0.556	0.516	53.94

On the full dataset, Logistic Regression achieved an accuracy of 0.51 and a macro-F1 of 0.49. After applying our dimensionality reduction pipeline—removing low-variance and low-importance features—the model improved to 0.56 accuracy and 0.52 F1, which might not be a significant improvement, however the runtime dropped by almost 90%.

While Logistic Regression did not reach the top performance of Random Forests, it handled the dataset's linear component well and provided interpretable insights into feature importance. However, the extremely long runtime (almost 400 minutes) highlights that linear solvers scale poorly with tens of thousands of features, especially under ElasticNet regularization. This makes Logistic Regression computationally demanding for high-dimensional text representations but still a strong and stable baseline. For comparison, random guessing with 50 target variables has an expected success rate of only 2%.

5.4 Model 2: Decision Tree

To prevent overfitting to noise we initially limited the growth of the tree by its parameters (`max_depth=15, min_samples_split=5`). However, this configuration produced the lowest performance among all models, so in a second run, we removed all depth restrictions (`max_depth=None`) to see if our chosen parameters caused the problems.

Allowing the tree to grow without depth restrictions slightly increased performance, but the overall improvement remained minor. Both configurations performed far below other models, indicating that the tree was unable to extract strong predictive patterns from the data.

Table 10: Performance of the Decision Tree (CV results).

Dataset	Mean Accuracy	Macro-F1
Reduced (depth = 15)	0.141	0.105
Full (depth = 15)	0.091	0.062
Reduced (no depth limit)	0.232	0.177
Full (no depth limit)	0.225	0.178

With 10,000 features and 750 samples, the feature-to-instance ratio severely limits the tree's ability to generalize. . Even after dimensionality reduction, the ratio between features and samples remained too large for a single tree to model effectively.

Consequently, while the model provided a useful diagnostic baseline for non-linear feature interactions, it was not suitable as a standalone classifier for this dataset.

5.5 Model 3: Support Vector Machine

Relevant parameters: `kernel="rbf"`, `C=2.0`, `gamma="scale"`

We trained a Support Vector Machine (SVM) with an RBF kernel to capture the dataset's non-linear relationships.

Table 11: Performance of SVM on full and reduced feature sets.

Dataset	Accuracy	Macro-F1	Runtime [min]
Full	0.332	0.303	0.52
Reduced	0.500	0.450	0.10

The SVM struggled on the full dataset, reaching only 0.33 accuracy and 0.30 macro-F1. This result was expected — with over 10,000 features but just 750 samples, the RBF kernel couldn't find a good boundary between classes.

After dimensionality reduction, performance improved substantially (accuracy 0.50, F1 0.45), and runtime dropped fivefold. This confirmed that feature filtering helped the SVM focus on the most informative subspace. However, even with these improvements, SVM still underperformed compared to Random Forests and ElasticNet Logistic Regression.

We found that the main limiting factors were the extremely high feature-to-sample ratio (13:1), which made kernel estimation unstable and the integer-only feature representation, which reduced the benefit of RBF's continuous mapping.

In summary, although the SVM captured some non-linear relationships, it was hindered by the dataset's high dimensionality and discrete feature space.

5.6 Other models

5.6.1 Random Forest

After the bad performnace of a single decision tree we also wanted to test a Random Forest.

Random Forests were trained as an ensemble of 300 decision trees. This method is better suited for datasets with high dimensionality and complex feature interactions, as it can capture non-linear dependencies without requiring extensive parameter tuning.

Dataset	Accuracy	Macro-F1	Runtime [min]
Full	**0.704**	**0.678**	**0.24**
Reduced	0.652	0.605	0.19

The Random Forest significantly outperformed all other single models tested on this dataset, achieving an accuracy of 0.70 and a macro-F1 of 0.68 on the full feature set. Notably, it reached this level of performance at a fraction of the runtime required by Logistic Regression (less than one minute versus over six hours).

Reducing the number of features slightly decreased performance, indicating that it hurt the Random Forest's ability to collect information from weakly contributing variables.

5.7 Summary

Table 12: Model benchmark results for full and reduced feature sets.

Model	Dataset	Accuracy	Macro-F1	Runtime [min]
LightGBM	Full	0.543	0.505	17.37
	Reduced	0.560	0.519	12.34
Logistic Regression	Full	0.513	0.490	397.58
	Reduced	0.556	0.516	53.94
SVM	Full	0.332	0.303	0.52
	Reduced	0.500	0.450	0.10
Random Forest	Full	**0.704**	**0.678**	**0.24**
	Reduced	0.652	0.605	0.19

Random Forest achieved the highest accuracy (0.70) and Macro-F1 (0.68) on the full dataset, making it the best overall performer. LightGBM, which was initially trained for features selection, performed moderately well (Accuracy 0.54, F1 0.50) but required much longer runtime ($\sim$17 minutes). SVM performed poorly on the full feature set (F1 $\approx$ 0.30).

On the reduced dataset, performance remained stable or improved slightly for most models, while runtimes decreased substantially. Logistic Regression and SVM benefited the most from reduced dimensionality, whereas Random Forest maintained strong performance, demonstrating that the feature selection pipeline preserved key predictive information.

6 Summary

Note: For models that were trained with different parameters the best result is shown in the table.

Dataset	Model	CV Accuracy	macro-F1
Credit Score	Logistic Regression	0.970 ± 0.024	0.964 ± 0.030
Credit Score	Decision Tree	0.980 ± 0.024	0.976 ± 0.029
Credit Score	SVM	0.880 ± 0.050	0.860 ± 0.060
Vehicle	Logistic Regression	0.850	0.850
Vehicle	Decision Tree	0.700	0.710
Vehicle	SVM	0.870	0.870
Vehicle	Random Forest	0.760	0.760
Vehicle	KNN (PCA)	0.780	0.780
Congressional Voting	Logistic Regression	0.968	0.959
Congressional Voting	Decision Tree	0.963	0.948
Congressional Voting	SVM	0.954	0.937
Congressional Voting	Naïve Bayes	0.917	0.895
Reviews	Logistic Regression	0.556	0.516
Reviews	Decision Tree	0.232	0.177
Reviews	SVM	0.500	0.450
Reviews	Random Forest	0.704	0.678

Table 13: Cross-validation results across datasets and models

Across all four datasets, the three selected classifiers (Logistic Regression, Decision Tree, and SVM) demonstrated complementary strengths that highlight the trade-offs between interpretability, flexibility, and generalization.

For the small **Credit Score dataset**, both Logistic Regression and the Decision Tree achieved near-perfect performance (`CV_Accuracy` ≈ 0.97–0.98), with the latter slightly higher but prone to overfitting due to small sample size and a dominant predictor. The SVM achieved lower yet stable performance (0.88), indicating better regularization and generalization potential.

In the **Vehicle dataset**, where the feature space was larger and multicollinear, Logistic Regression and SVM remained robust (0.85–0.87), whereas tree-based and instance-based models (Random Forest, KNN) underperformed, especially without dimensionality reduction. PCA modestly improved KNN results by $\approx 5\%$, but interpretability was lost.

For the **Congressional Voting dataset**, all models performed exceptionally well, with Logistic Regression (L1) and the pruned Decision Tree both exceeding `CV_Accuracy` = 0.96. Despite marginally lower accuracy, the penalized Logistic Regression was preferred due to its stability and interpretability. Naïve Bayes, although conceptually appropriate, lagged behind at 0.91.

Finally, in the **Amazon Reviews dataset**, characterized by extreme feature dimensionality (10,002 predictors for 750 samples), ensemble and regularized models dominated. The Random Forest achieved the highest performance (`CV_Accuracy` = 0.704, `Macro-F1` = 0.678), confirming its suitability for wide, sparse, and potentially noisy data. The ElasticNet-based SGD model performed competitively (0.685 / 0.653), suggesting that linear methods still retain value even in high-dimensional settings when properly regularized.

Overall, the Decision Tree and Random Forest models tended to overfit small or unbalanced datasets, while Logistic Regression and SVM offered more consistent generalization across diverse feature structures. This emphasizes that no single model is universally optimal; instead, robustness and interpretability vary with dataset size, balance, and dimensionality.

References

[1] Kaggle. *184-702 TU ML 2025 W: Congressional Voting*. Accessed November 9, 2025. 2025. URL: https://www.kaggle.com/competitions/184-702-tu-ml-2025-w-congressional-voting.

[2] Kaggle. *184-702 TU ML 2025 W: Reviews*. Accessed November 9, 2025. 2025. URL: https://www.kaggle.com/competitions/184-702-tu-ml-2025-w-reviews.

[3] OpenML. *analcatdata_creditscore*. 2025. URL: https://www.openml.org/d/461.

[4] OpenML. *Vehicle Dataset*. Accessed November 9, 2025. 2025. URL: https://www.openml.org/search?type=data&sort=runs&status=active&id=54.

[5] Jeffrey S. Simonoff. *Analyzing Categorical Data*. New York: Springer-Verlag, 2003.