Customer Feedback Classification Model

Abraham Assariparambil Earnest (z5481911)

Devam Jain (z5457085)

Dhwanish Kshatriya (z5421168)

Nicholas Langford (z5487536)

Rohan Agarwal (z5423480)

Table of Contents

Introduction 2
Preliminary Investigation 2
Evaluation Metrics 2
Class Imbalance, Prevalence & Impact 2
Relationships in the Data (Feature-Feature & Feature-Response) 3
Methodology 4
Results & Discussion 5
Distribution Shift Analysis & Resolutions 5
Conclusion 6
Appendix 1 – Plots & Figures 7
Appendix 2 – Data Balancing Further Literature Review 19
Glossary 22
References 23

Introduction

The following report explores the analysis, development and evaluation of a multi-class text classification model. The problem statement aims to categorise vector embeddings of customer feedback into 28 distinct product categories. Throughout the report, we investigate problems surrounding severe class imbalances, a high dimensional feature space and distribution shift in production data. To handle the imbalances, we employed cost-sensitive methods in our logistic regression model, and SOTA sampling techniques such as SMOTE and bagging for our random forest model. Despite the high dimensionality, features were shown to be highly informative through permutation importance and thus were selected with caution. Testing our production data revealed a label shift, specifically that the marginal distribution of labels changed from source to deployment, while the difference between multivariate feature distributions remained constant.

Preliminary Investigation

When performing basic integrity checks, we saw no immediate issues such as null values or duplicates. Accordingly, the data was split into a training and holdout set using a 25% split (with stratification). Looking at the distribution of means and variances in Figure 1, we observe that our covariates are roughly standardises with means between -0.12 and 0.12, and standard deviations around 0.96 and 1.04. However, when performing a D'Agostino-Peason's test, we observe that only one feature conforms to normality. After clipping beyond 2.5 standard deviations from the mean, we see that 101 features conform to normality, suggesting that the assumption is not unreasonable. Then, after defining outliers as observations beyond 3 standard deviations from the mean, we see that there are fewer than 75 outliers per feature, but over 38% of the rows with outliers. Performing a t-test and Levene's test at a 95% confidence level, we see that these outliers do not have a statistically significant impact on the mean of any features, but they do have a statistically significant impact on the variance of all features. This may potentially affect the performance of variance-sensitive techniques.

Evaluation Metrics

Accuracy measures the proportion of predictions that were correct. This makes it highly susceptible to biasing towards the majority class. For instance, if a few classes dominate a dataset, a model could achieve high accuracy by correctly classifying the majority classes while failing to identify correctly for rare classes. This does not reflect our model's true requirements, which involves achieving a high performance for both majority and minority classes. Thus, accuracy is a poor measure of the model's overall performance. Instead, weighted cross entropy (WCL) can be used. WCL is a probabilistic measure that can assign weights and thus different penalties for misclassification on different classes. Typically, the weights are calculated as the inverse of class prevalence in the dataset. In doing so, this ensures that a high performing model has "learned from the underrepresented classes too", rather than focussing only on dominant ones (Farhadpour et al., 2024). When evaluating the model's per-class performance, we can use precision and recall. Precision measures what proportion of a model's predictions of class X are in class X. A low precision in class X indicates that it being predicted incorrectly too frequently leading to too many false positives – an issue for majority classes which we aim to mitigate. Alternatively, recall measures what proportion of class X has been predicted as class X. A low recall in class X indicates that it is predicted correctly too infrequently, resulting in a too few true positives – an issue for minority classes which we aim to mitigate.

Class Imbalance, Prevalence & Impact

By analysing the frequency of each class in our training data, we see that there is a significant imbalance. Figure 5 showcases the severity of this imbalance, with a count of 3359 and 4 for our majority and minority classes respectively. This corresponds with an imbalance ratio of 839.75, which can result in a significant decline in model performance. We observe many issues 11/28 classes having less than 50 training instances, with classes 1, 2, 16 and 22 being particularly problematic with under 10 instances each (less than 0.07% of the data each). This can result in a reduced sensitivity towards feature relationships amongst the minority classes, and thus a poor classification performance. Furthermore, we also observe 3359 observations for the majority class (class 5, with over 40% of the training data), potentially introducing model biases towards the majority class. By training and evaluating a naïve model an unmodified version of the data, we can directly see the implications of the imbalance. For reference, the model used was a default random forest classifier with a max depth of 8. The model's confusion matrices in Figure 6 showcases that the majority class (5) was predicted very frequently, even when it was not the true label. This indicates that the model may be biasing towards predicting the majority due to its frequency in training. Further, the validation classification report in Figure 7 showcases a precision and recall of minority classes (such as 1, 2, 16 and 22). This showcases that the model fails to learn the relationships between the features and responses for minority classes.

The first data balancing method is cost sensitive methods. Cost sensitive approaches work by considering higher costs for the misclassification of minority samples with respect to majority samples. This in turn enhances model performance on imbalance datasets without requiring extensive preprocessing (He & Garcia, 2009). Additionally, cost-sensitive methods are flexible and can be integrated into various machine learning algorithms, including neural networks, and support vector machines, making them versatile for different tasks (Zhou & Liu, 2006). Thus, cost sensitive methods are suitable for our high-dimensional multi-class dataset by emphasising minority class accuracy and simplifying complex category mapping. Next is oversampling. Over-sampling is a data balancing method which increases the size of the minority class to balance the majority class, by either duplicating existing data or generating new data (Akila, 2016). One method is random oversampling, which duplicates existing samples. However, this can risk overfitting and reinforcing biases without adding new information (Akila, 2016; Chawla et al.). Another method is SMOTE, a synthetic oversampling technique. SMOTE works by generating samples along line segments connecting k-nearest neighbours of minority class instances, offering more diverse data (Rendón et al., 2020). However, SMOTE can introduce noisy data points or complicate decision boundaries, especially in our high-dimensional multi-class dataset, leading to overfitting and reduced generalisation (He-ieee, 2008). Finally, there is under sampling. Under-sampling is a data balancing technique which involves decreasing the size of the majority class to balance the minority classes, reducing computational burden (He, 2013). However, since random under sampling discards majority examples arbitrarily, it could result in data loss, potentially missing useful information required for accurate modelling. Therefore, balanced bagging is recommended. This is an ensemble method which trains base learners on a dataset where all minority samples are included, and the majority class is partitioned into equal-sized disjoint subsets. This is to keep balance across bootstraps, reduce information loss and improve minority class recall. For our dataset, it is particularly effective due to its ability to handle noisy data and high-variance learners, as it reuses majority class subsets systematically across bootstraps. More can be read about the exact algorithms behind these techniques in Section 2, Section 3 and Section 4 in the Appendix.

Relationships in the Data (Feature-Feature & Feature-Response)

To analyse the relationships between features, we looked first looked at both Pearson (linear) and Spearman's (rank) correlation. We observe that there are no pairs of features with an absolute correlation above 0.7, suggesting there is no strong correlation in our data. Thus, there is a lower risk of multicollinearity. This aligns with the theoretical behaviour of text embeddings, which are typically trained to embed "almost orthogonal" vectors to minimise redundancy across dimensions and maximise semantic expressiveness (Andrecut, 2018, p.3). To further investigate the redundancy in the data, we plotted the cumulative explained variance by the number of principal components in Figure 8. Due to the statistically significant impact of outliers on the variance of our features, the outliers were clipped to 3 standard deviations. Figure 8 showcases that it requires 176 principal components to explain 95% of the variance in the data, and 125 principal components to explain 90%. Thus, it takes requires over 41% of the total number of components to explain 90% of the variance, supporting the relatively moderate or lower feature redundancy in our data. Next, we looked at the feature-response relationships in our data. Due to the relatively lower levels of collinearity in our data, we decided it was valid to use permutation importance to measure the relative contribution of covariates in model performance. In Figure 9, we see that even the top features have a very small magnitude of decrease in weighted cross entropy (when permuted). This indicates that the models may not need to be sensitive to a single feature. This is reinforced by the large error bars, indicating large fluctuations in importance scores. Overall, the results show that the top 5 features by permutation importance are 213, 143, 256, 255, and 214 (descending). When extending this to look at all features, Figure10 showcases how there were 64 features with a negative mean importance score. However, their low magnitude of (negative) importance and relatively large error bars makes it difficult to conclude whether those features are truly harming the model (when permuted). Further, looked at the absolute value of coefficients of a one-vs-rest logistic regression with weighted cross entropy to evaluate the impact of features in a per-class basis. Figure 11 shows us that across all 28 classes, feature 65 can be seen in the top 5 features by absolute coefficient only 3 times. All other features in the top 5 are only present in the top 5 for 1 or 2 classes. This reaffirms the idea that no single feature showcases standalone importance in modelling the response. Looking at the coefficients for classes with less than 10 training instances in Figure 11, we see that this notion holds true even for the significantly underrepresented classes.

Methodology

Considering the high dimensionality and relatively low training instances in our dataset, highly complex deep learning models such as feedforward neural networks were deemed unfit, as further explored in Section 1. Accordingly, the shape of the dataset, relatively low feature redundancy and relatively low correlation values (both Pearson and Spearman) made the logistic regression a strong fit. This is due to their weaker assumptions surrounding multicollinearity and feature dependence and ability to handle higher dimensional data with regularisation. Initially, a weighted cross entropy function was used in training to account for the class imbalanced in the data. However, after observing low precision and recall in the minority classes, an additional class_weight='balanced' was used to further shift learning from majority to minority classes. To further focalise learning on minority classes, a one-vs-rest variant of the logistic regression was preferred. However, as previously discussed the covariates lacked adherence to normality (especially in their tails) and displayed a statistically significant impact of outliers on their correlation. To account for this, features were clipped at 2.5 standard deviations to reduce the impact of these outliers and increase adherence to normality when using the logistic regression-based models. Given that our feature-feature relationships showed no significant signs of redundancy, and our feature-response relationships did not lead to any confident conclusions, we decided not to perform any feature selection for our logistic regression model. Instead, L2 (RIDGE) regularisation to mitigate the risk of overfitting. This was preferred over L1 (LASSO) due to its smooth and convex objective which leads to a faster and more stable convergence. Due to the large number of features, it was intuitive that a relatively large magnitude of shrinkage (small C) would need to be applied. Therefore, we used 4-fold stratified cross validation to test 50 equidistant values of C between 10^{-5} and 10, from which $C = 4.894 \times 10^{-3}$ resulted in an average validation weighted cross entropy of 0.00489. This was better than all alternative models and thus was selected for our predictions.

Whilst the logistic regression offered a robust baseline, confusion analysis showed that we may be able to benefit from a model that can capture more complex decision boundaries. A natural consideration was a decision tree-based model, which are typically lower in bias risk instability. To mitigate this, we implemented ensemble techniques (bagging) to form a random forest classifier which can achieve these complex decision boundaries and thus lower bias whilst simultaneously reducing the variance compared to a typical decision tree. Specifically, a BalancedRandomForestClassifier (which under samples using bagging) was selected to handle the class imbalance. As classes except were under sampled to the minority size (of only 4 in training), we used SMOTE to oversample all underrepresented classes to a minimum size of 200. This prevented information loss in training for complex decision boundaries. Initially, all classes were under sampled, however after observing that the majority class was frequently confused with class 14, we decided to under sample all classes that are 'not majority' to prevent the information loss. To account for the consequential imbalance in the data, a 'balanced_subsample' class weight was applied. The 61 features with a negative permutation importance were removed as a part of feature selection, primarily to reduce computational cost. Initially, principal component analysis was used to further reduce the dimensionality of the data and improve the quality of synthetic oversampling (Mulla, Demis & Hassan, 2021). However, this resulted in significant information loss and thus was not applied in the pipeline for our final model. Entropy was used in training instead of Gini due to its increased sensitivity to minority classes. To ensure robustness, n_estimators was set to 500. Then, GridSearchCV with 4 stratified folds was used to identify the hyperparameters that prevent overfitting. These were found to be {'max_depth': 7, 'min_samples_split': 3, 'min_samples_leaf': 1, 'max_features': 'sqrt', 'class_weight': 'balanced_subsample'}. This model received an average cross validation weighted log loss of 0.00711.

Results & Discussion

Recall that our data was initially separated into a train and test set using a 25% split (with stratification). After hyper parameter tuning using CV, the models were trained on the entire training data (7500 observations). The logistic regression model received a test weighted log loss of 0.00523, which is a slight decline over its average CV loss of 0.00489. Looking at the performance from a per-class basis in Figure 13, we see that the majority class had a precision of 0.96 and recall of 0.65. This suggests that the model is only predicting class 5 when it has high confidence, resulting in many false negatives. This is likely due to the additional penalty (WCE) it faces for getting smaller classes incorrect in training (where weights are roughly squared inverse frequency of a class). From Figure 13 we also see that the majority is often misclassified as class 14, suggesting there may have been a complex decision boundary that the logistic regression was not able to capture effectively. Looking at the minority classes, we see that this additional penalty in training has in an increased precision and recall for classes 1, 16 and 22 compared to earlier baseline models. They showcase an F1 score of 0.5, 0.18 and 0.5 respectively, indicating that whilst these classes are being found, they are also being over predicted. This indicates that whilst the harsh penalty assists the learning process of minority classes, it is also harming the learning process of other classes. This is reinforced by the moderate macro average recall of all classes (0.58). Thus, whilst the model improves significantly compared to Naïve baselines, it leaves much to be desired in its learning process. Going forward, improvements could be made by using models that capture more complex decision boundaries and using hybrid approaches between oversampling and cost-sensitive training to mitigate the issues above.

The BRF model was implemented to test this hybrid approach, but its performance was much weaker. Table 1 shows the model's evaluation metrics, with a weighted log loss of 0.00711 and 0.00835 on validation (average of 4 folds) and test sets respectively. This is likely due the harsh sampling, with large classes (except for the majority) sampled down to a size of 200, potentially leading to information loss. It could also be attributed to the synthetic oversampling of minority classes to a size of 200, potentially introducing noise in the model's learning process. We also observe a complete failure to model and predict the minority classes in Figure 14 with each showcasing a precision and recall of 0. Despite this, we see a significant improvement in the recall of class 0, 7, 9, 11 and 15, indicating that the pipeline may be improving from the reduced sensitivity brought by the approach in the logistic regression. Overall, the performance of this approach was extremely poor, but going forward it could be improved by collecting more data for minority classes to allow for more accurate synthetic oversampling and thus less noise in the data.

Distribution Shift Analysis & Resolutions

From the histogram in Fig. 15, we clearly see the marginal label distribution has altered in production. Now The KS test cannot be applied in two or more dimensions (Feigelson and Babu, Center for Astrostatistics, Penn State University). Hence, we use Maximum Mean Discrepancy (MMD) via the alibi-detect library (Van

Looveren et al.) (see distribution_shift_eda.py) to measure the difference between multivariate feature distributions in a kernel embedding space (See Fig 18.). The high p-values indicate that for most class labels the distribution of the features given a particular class is from the same distribution. Hence this is highly indicative of **label shift**. Next, we compare the multivariate feature distribution in training and deployment, this can again be done using a simpler invocation of MMDDrift (see distribution_shift_eda.py), and from Fig. 19, we see that they do differ – the first condition for covariate shift is satisfied. Next, we compare the distribution of labels given inputs between training and deployment by comparing weighted log loss of a model learned on training data. From Fig 20. (extracted from distribution_shift_correction.py), we see a substantial difference in the weighted log losses, despite a similar f1-score. Due to the class imbalance the log loss difference is more significant and hence this implies that the second condition for covariate shift is not satisfied.
Traditional machine learning models assume that the training and deployment data are i.i.d. If violated by distribution shift, learned patterns may not generalise well to new data. Several methods exist to mitigate this such as Data Augmentation, Domain Adaptation and Importance Weighting (See Glossary for details). Below describes the use of importance weighting as one of the methods to mitigate distribution shift under the given project constraints. Importance weighting was performed to match the target joint distribution, adjusting the training loss to emphasise samples in a way that mimics the target label distribution, thus correcting for label shift. The weighted log loss of a model improved from 0.016 to 0.0156 by modifying sample weights, using the optimal weight of the inverse class frequency in the source (training) distributions. See Fig 21. and Fig 22. for further details, as well as distribution_shift_correction.py.

Conclusion

This project involved the analysis and modelling of a multi-class text classification dataset. Our objective was to develop a model capable of classifying customer feedback into 28 categories whilst simultaneously accounting for the high dimensionality of vector embeddings, severe data imbalances and distribution shifts in production data. A weighted logistic regression model with class balancing and regularisation proved effective, achieving a test weighted log loss of 0.00523. While precision and recall improved for several minority classes, the model struggled with complex decision boundaries and class overlap, particularly between the majority class and class 14. A follow-up Balanced Random Forest model incorporating SMOTE and under sampling performed worse, likely due to excessive sampling-induced noise and information loss. Although attempts were made to mitigate covariate and label shift via importance weighting, distribution mismatch remained a challenge. With more time, improvements could be made by collecting more data for underrepresented classes, fine-tuning hybrid sampling strategies, and employing deep learning models with domain adaptation techniques to better capture distributional complexities in production environments.

Appendix 1 – Plots & Figures

Figure 1 - Distribution of Feature Means

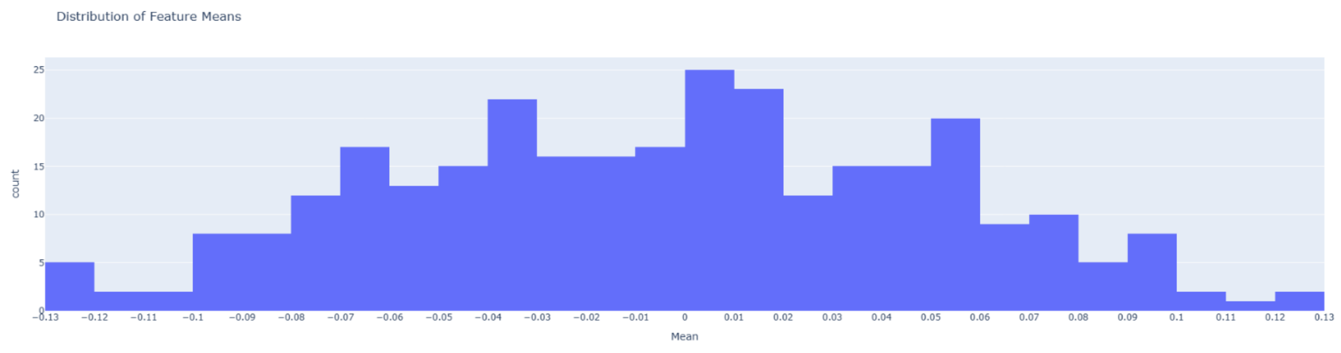


Figure 2 - Distribution of Feature Standard Deviations

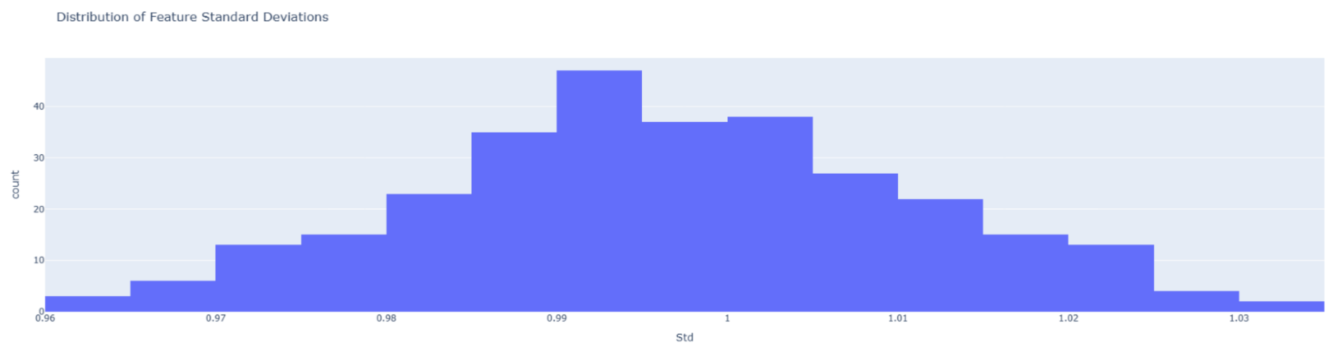


Figure 3 - Distribution of Outliers by Feature

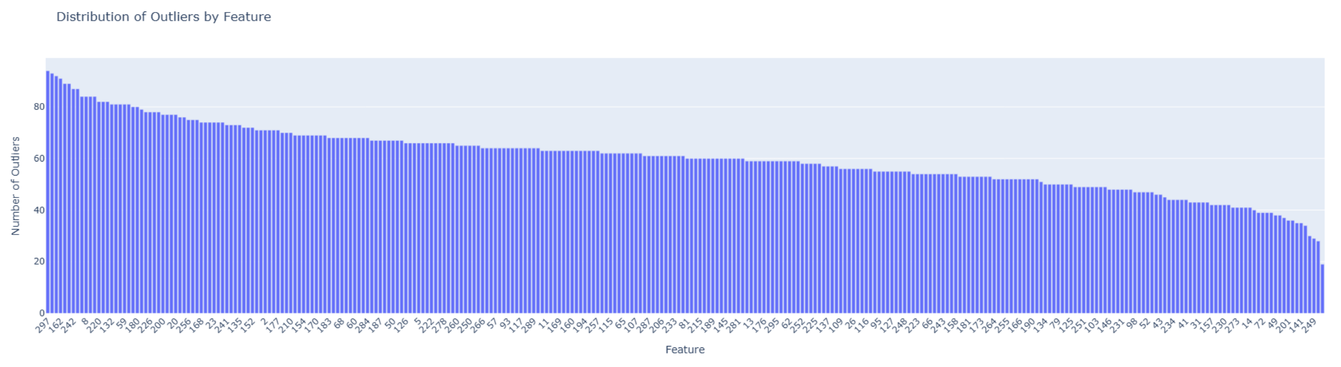


Figure 4 – Distribution of Levene Statistic by Feature

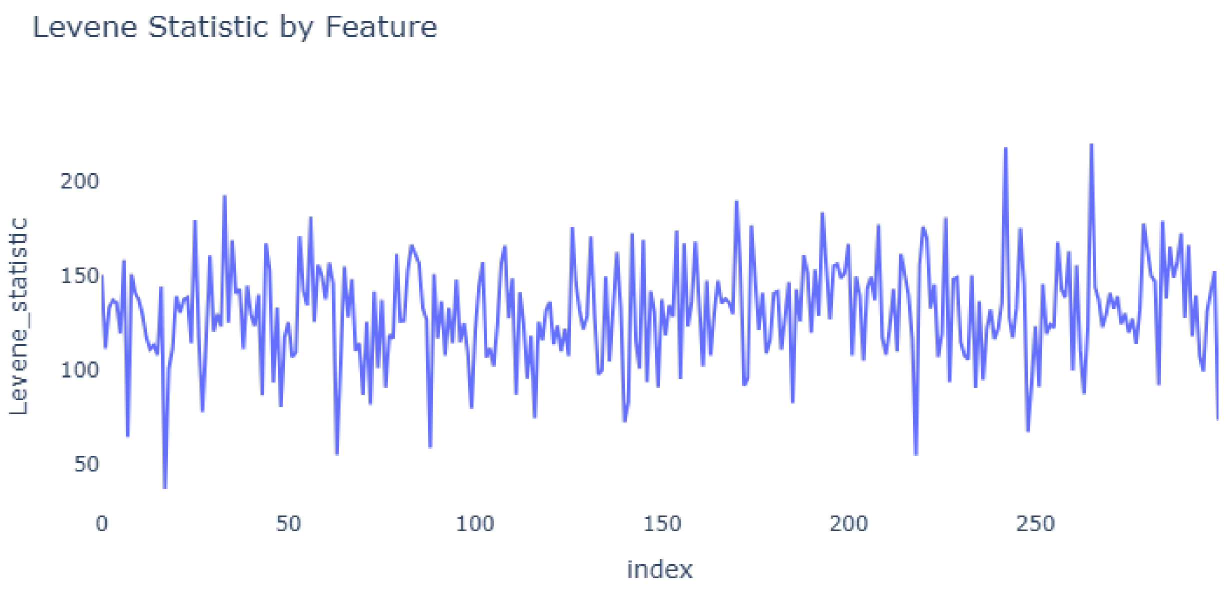


Figure 5 - Distribution of Response Classes

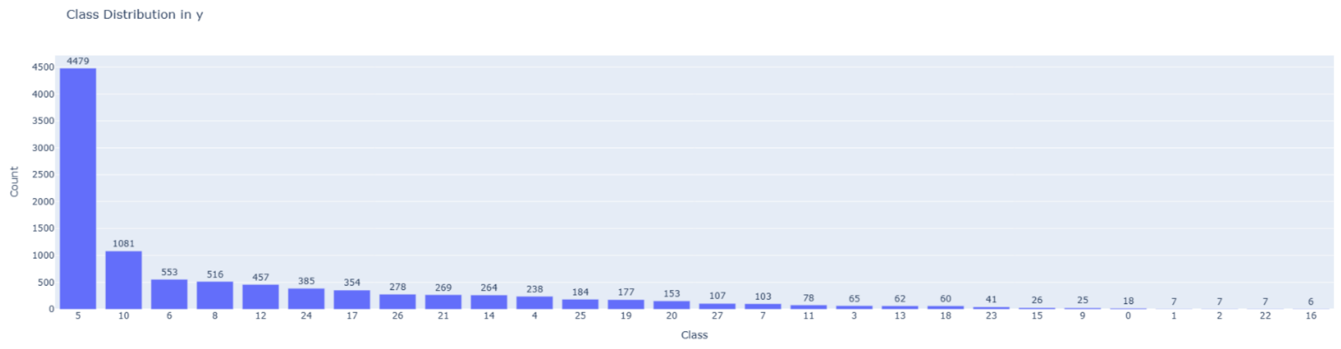
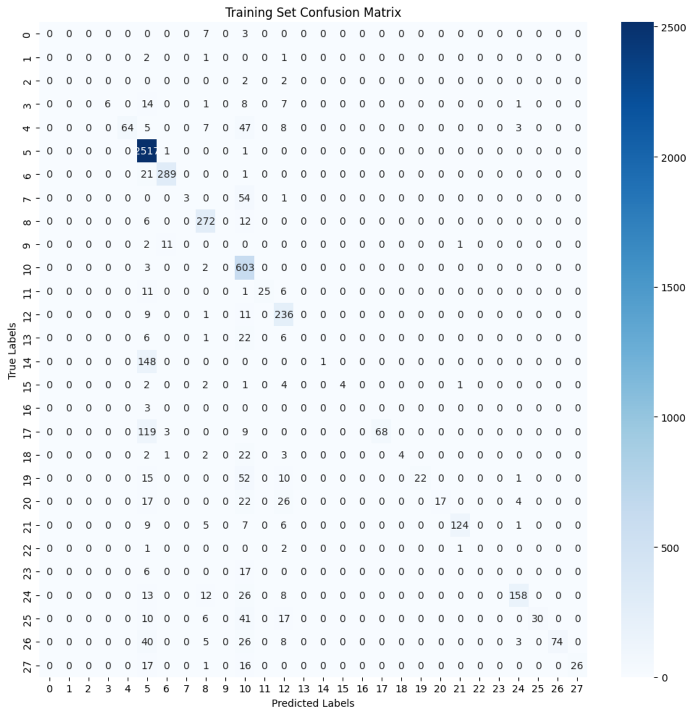


Figure 6 - Impact of Imbalance on Naive Model Confusion Matrix

The model used was a (sklearn) default random forest classifier with max_depth=8.



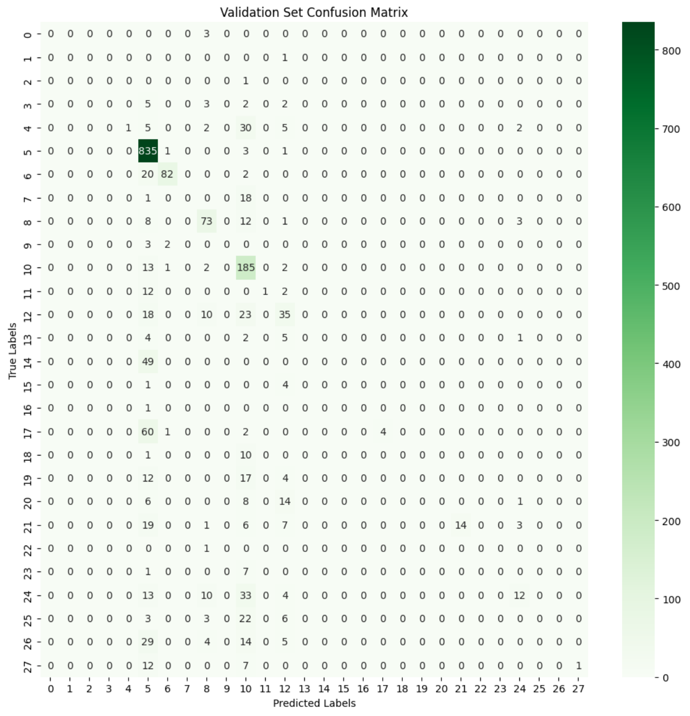
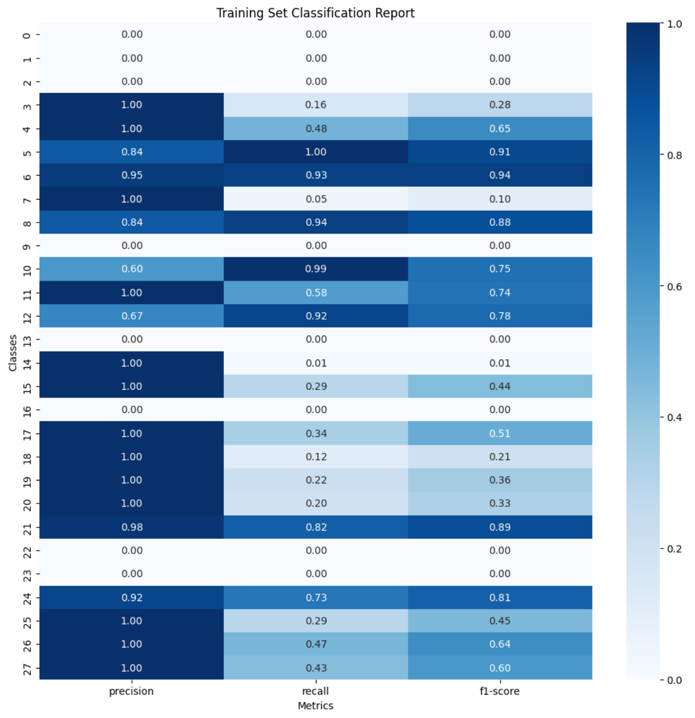


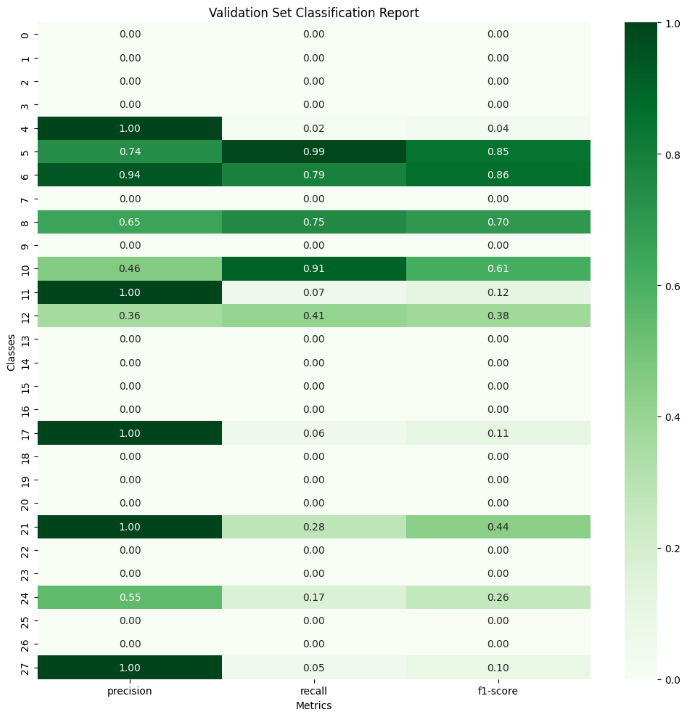
Figure 7 - Impact of Imbalance on Naive Model Classification Report

The model used was a (sklearn) default random forest classifier with max_depth=8.


Training Set Classification Report

Classes	precision	recall	f1-score
0	0.00	0.00	0.00
1	0.00	0.00	0.00
2	0.00	0.00	0.00
3	1.00	0.16	0.28
4	1.00	0.48	0.65
5	0.84	1.00	0.91
6	0.95	0.93	0.94
7	1.00	0.05	0.10
8	0.84	0.94	0.88
9	0.00	0.00	0.00
10	0.60	0.99	0.75
11	1.00	0.58	0.74
12	0.67	0.92	0.78
13	0.00	0.00	0.00
14	1.00	0.01	0.01
15	1.00	0.29	0.44
16	0.00	0.00	0.00
17	1.00	0.34	0.51
18	1.00	0.12	0.21
19	1.00	0.22	0.36
20	1.00	0.20	0.33
21	0.98	0.82	0.89
22	0.00	0.00	0.00
23	0.00	0.00	0.00
24	0.92	0.73	0.81
25	1.00	0.29	0.45
26	1.00	0.47	0.64
27	1.00	0.43	0.60

Metrics


Validation Set Classification Report

Classes	precision	recall	f1-score
0	0.00	0.00	0.00
1	0.00	0.00	0.00
2	0.00	0.00	0.00
3	0.00	0.00	0.00
4	1.00	0.02	0.04
5	0.74	0.99	0.85
6	0.94	0.79	0.86
7	0.00	0.00	0.00
8	0.65	0.75	0.70
9	0.00	0.00	0.00
10	0.46	0.91	0.61
11	1.00	0.07	0.12
12	0.36	0.41	0.38
13	0.00	0.00	0.00
14	0.00	0.00	0.00
15	0.00	0.00	0.00
16	0.00	0.00	0.00
17	1.00	0.06	0.11
18	0.00	0.00	0.00
19	0.00	0.00	0.00
20	0.00	0.00	0.00
21	1.00	0.28	0.44
22	0.00	0.00	0.00
23	0.00	0.00	0.00
24	0.55	0.17	0.26
25	0.00	0.00	0.00
26	0.00	0.00	0.00
27	1.00	0.05	0.10

Metrics

Figure 8 - Cumulative Explained Variance by Number of PC

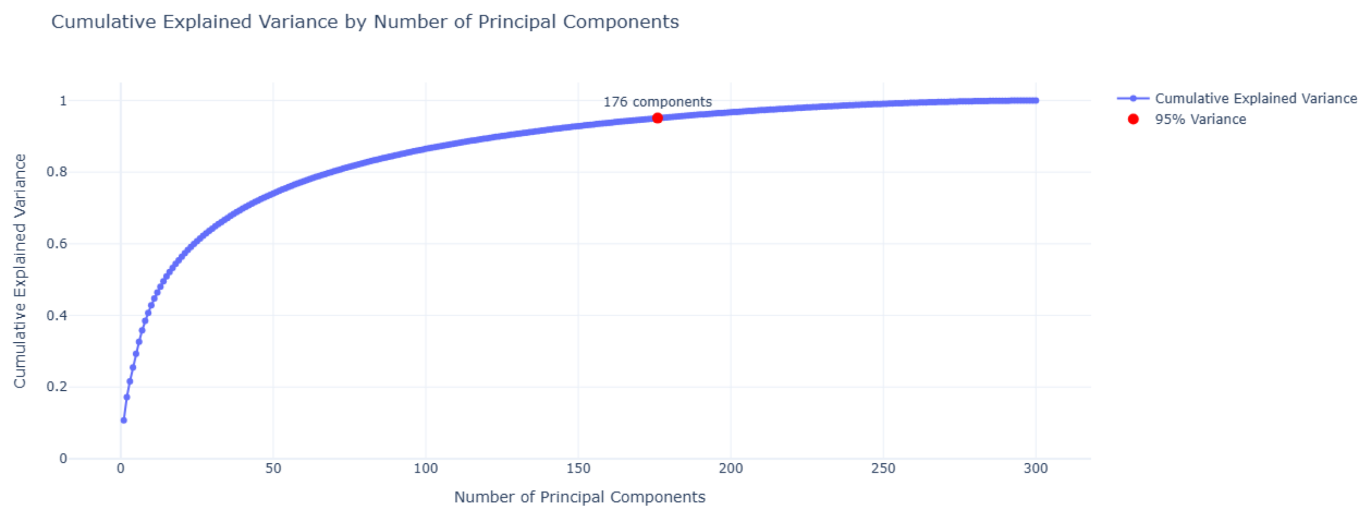
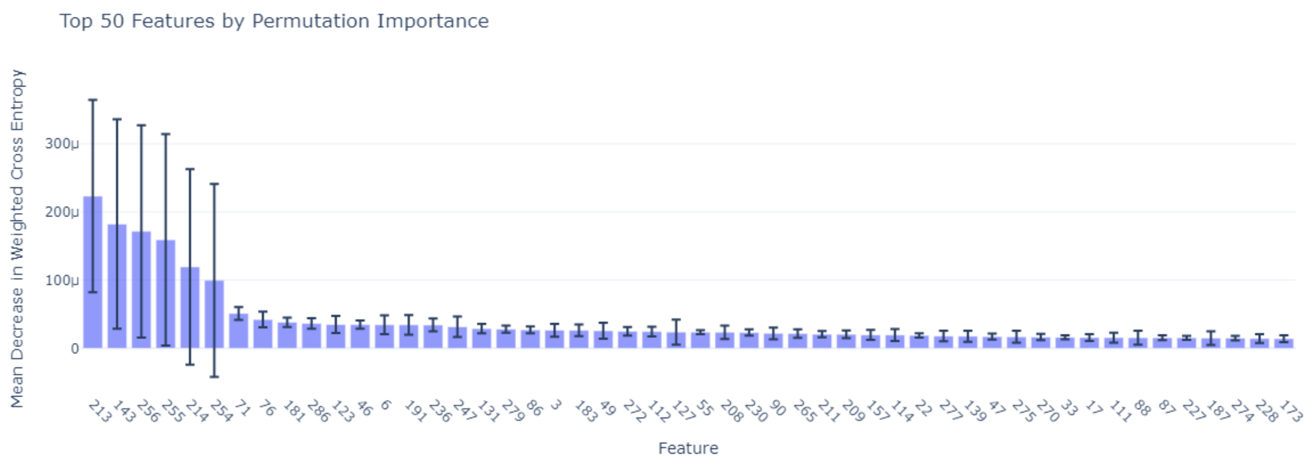


Figure 9 - Top 50 Features by Permutation Importance



Note, the following model was used as the estimator for the permutation importance.

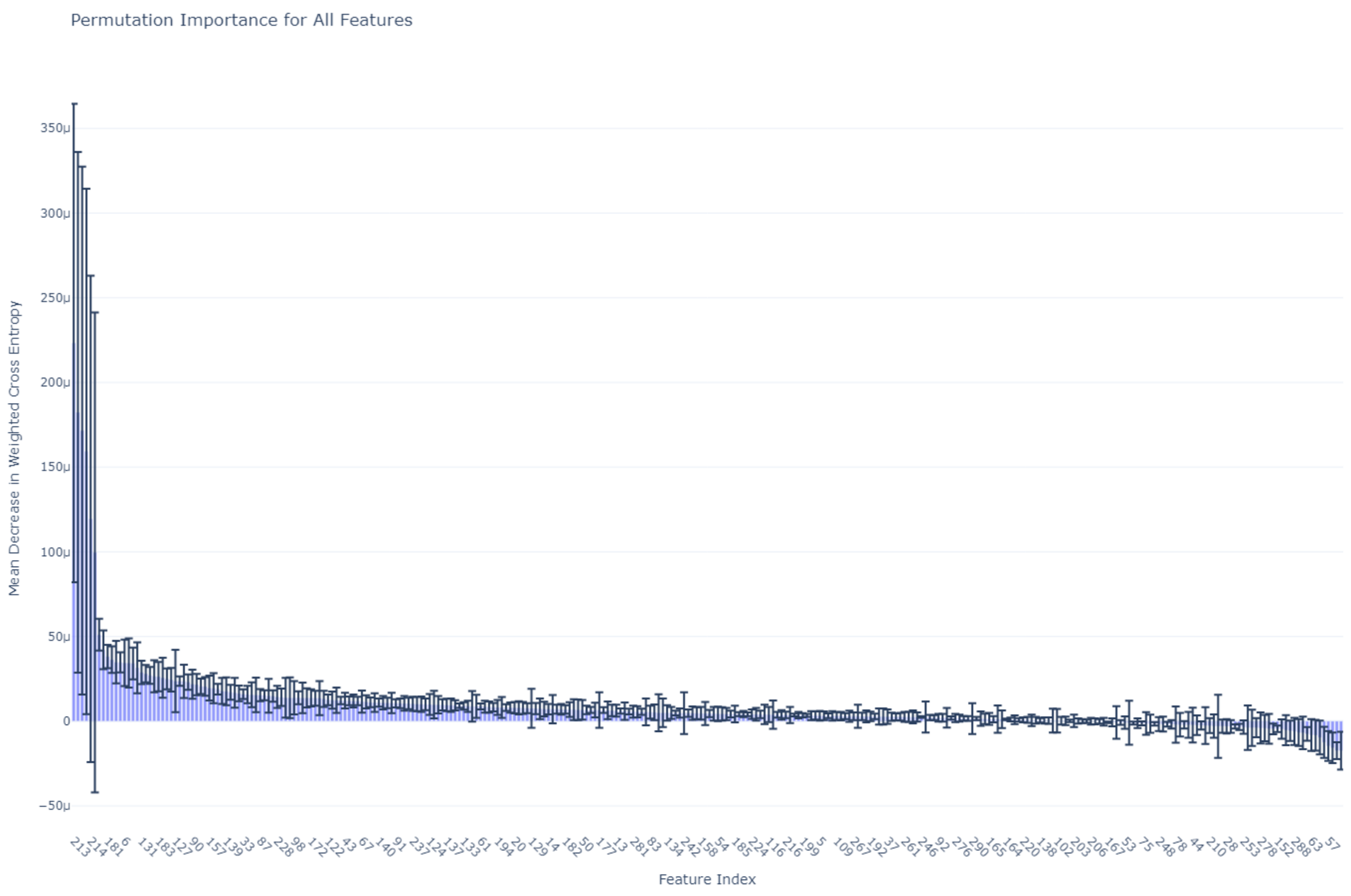
brf = BalancedRandomForestClassifier(n_estimators=100, criterion='entropy', max_depth=7, min_samples_leaf=1, min_samples_split=3, class_weight='balanced_subsample', oob_score=False,

random_state=0, n_jobs=-1).

Here, the 'class weight' was set to manage the class imbalance in training, and the 'max depth' and 'min samples split' were set to avoid overfitting. This, combined with 'num estimators' ensured stability of the underlying model.

A weighted cross entropy scoring function was used to ensure the measured performance accounts for data imbalances.

Figure 10 - Permutation Importance of Features



This shows 85 features with an average negative permutation importance over 10 iterations (removing them was beneficial for the model). Namely, these were:

['1', '2', '19', '24', '28', '29', '34', '38', '40', '44', '48', '53', '56', '57', '58', '59', '63', '68', '70', '75', '77', '78', '102', '104', '113', '115', '126', '135', '147', '152', '155', '156', '163', '166', '167', '176', '188', '189', '190', '203', '204', '206', '207', '210', '215', '218', '221', '223', '231', '233', '234', '239', '248', '251', '253', '257', '258', '269', '278', '288', '291', '296', '298', '299']

Note, the same model and metrics was used as in Figure 9.

Figure 11 - Frequency of Features in Top 5 of Any Class in OVR LogReg

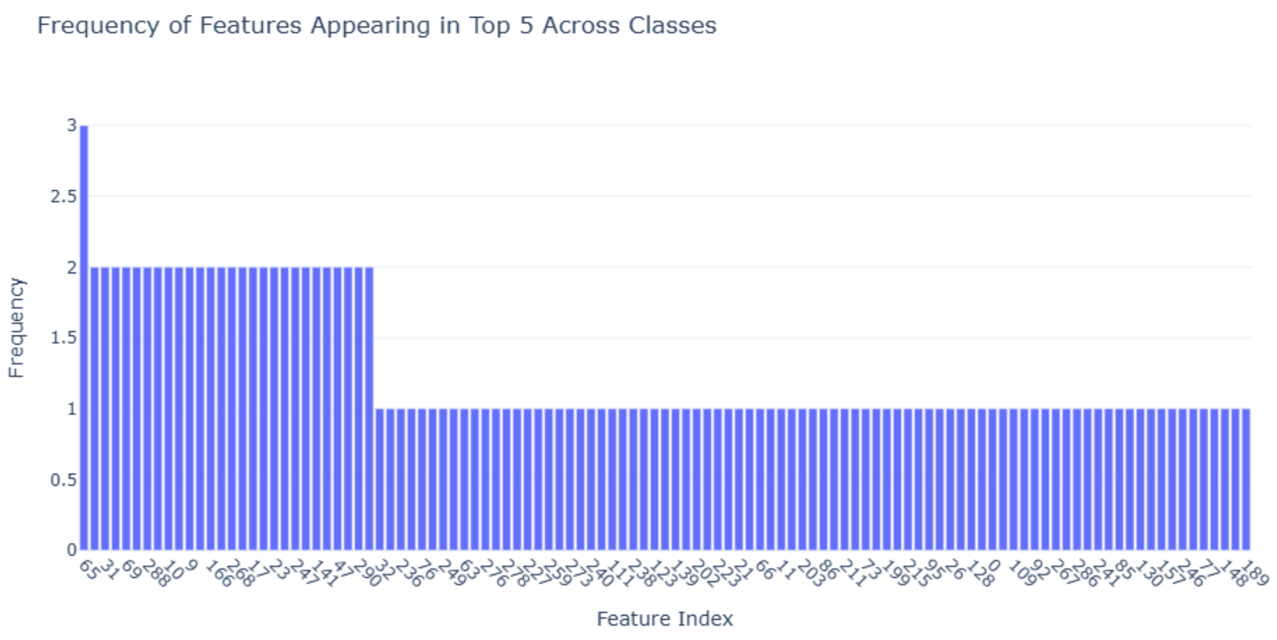
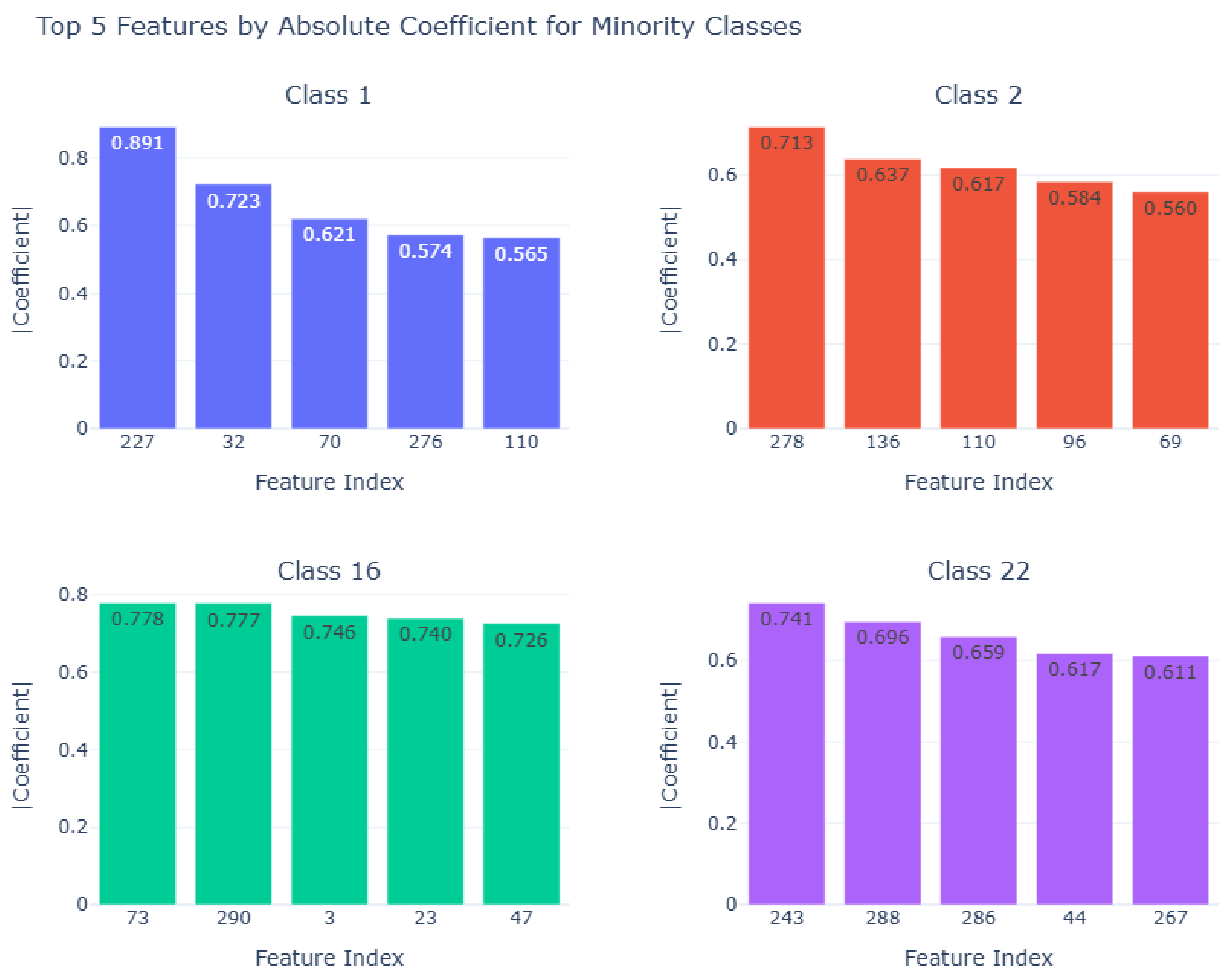


Figure 12 - Frequency of Features Appearing in Top 5 Across Minority Classes



Equation 1 - The PAC Bound

The generalization error $e(h)$ of a classifier can be bounded with high probability $1 - \delta$ as:

$$e(h) - \hat{e}(h) \leq \sqrt{\frac{1}{2n}\left(\log|H| + \log\left(\frac{2}{\delta}\right)\right)},$$

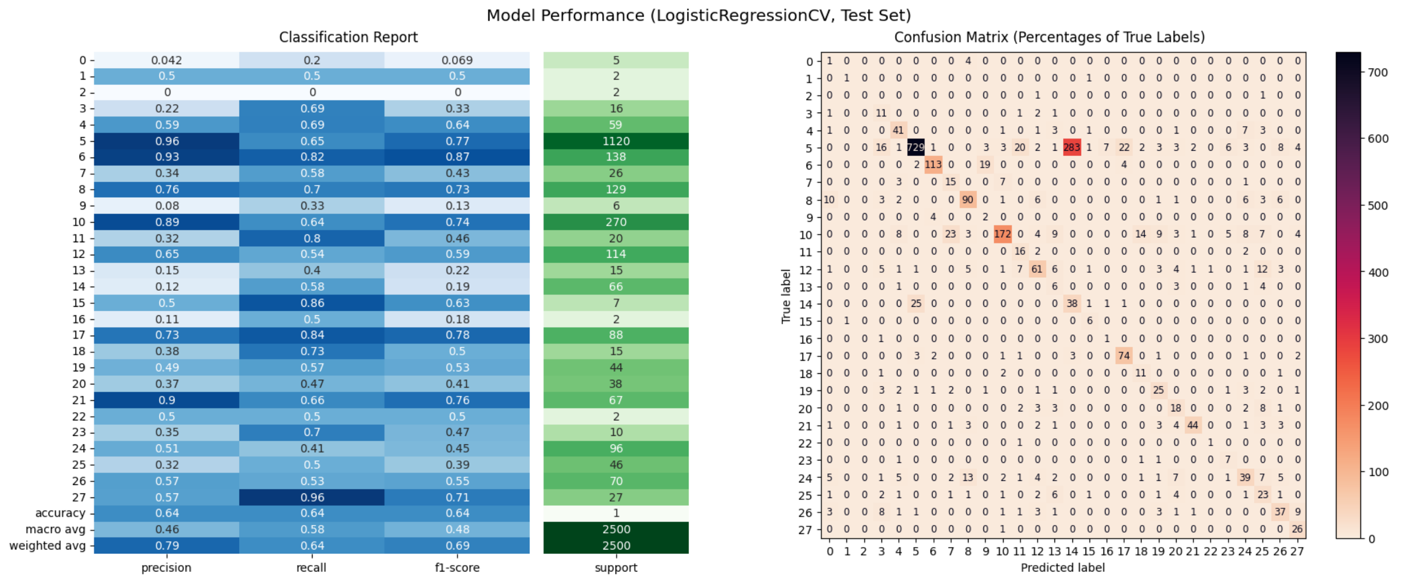
Where:

- $e(h)$ is the true error of the classifier h,
- $\hat{e}$ is the empirical error (error on the training set),
- $|H|$ is the size of the hypothesis space (the number of possible classifiers),
- n is the number of training samples,
- δ is the confidence parameter (with probability $1 - \delta$).

Section 1 - Risks with Neural Networks

Neural networks have proven to be highly capable in solving the multi-class text classification problem. However, due to the complexity of their hypothesis space, the number of parameters can lead to a large generalization error – an issue that is well documented in literature, with the PAC bound (see Equation 1, Kouw & Loog, M. (2019)). Such notions were reciprocated in out empirical testing, with attempts at capturing complex relationships resulting in instability of our feedforward neural network.

Figure 13 - OvR Logistic Regression Results (Test Set)



Model Performance (LogisticRegressionCV, Test Set)

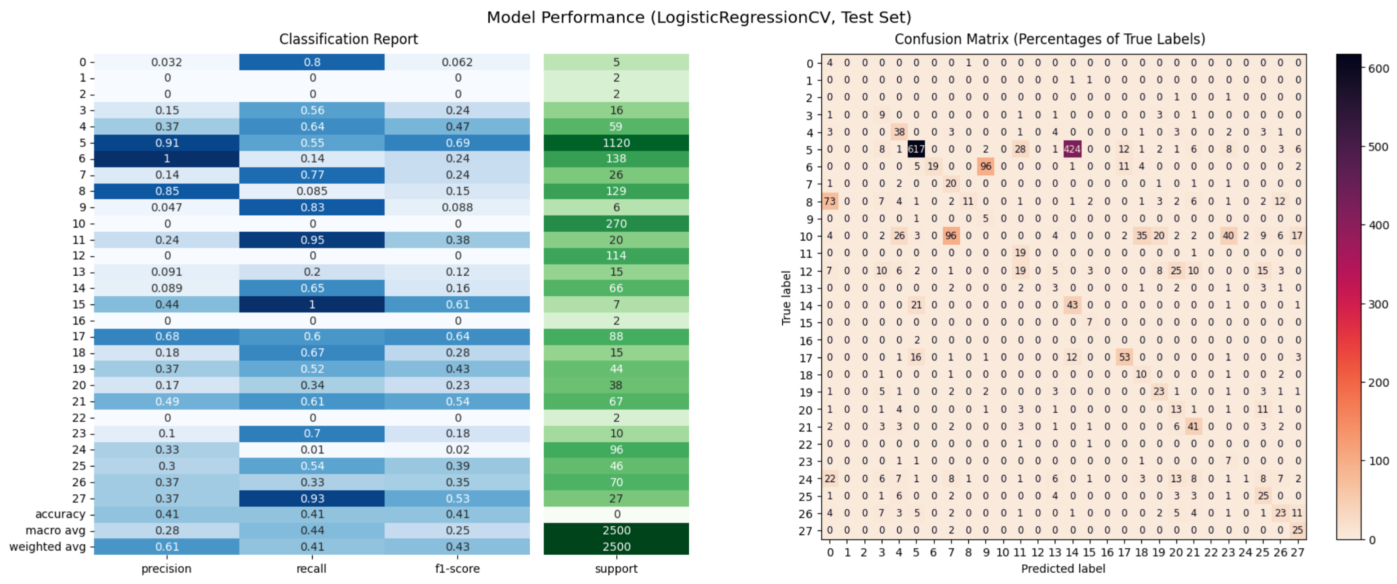
Classification Report

	precision	recall	f1-score	support
0	0.042	0.2	0.069	5
1	0.5	0.5	0.5	2
2	0	0	0	2
3	0.22	0.69	0.33	16
4	0.59	0.69	0.64	59
5	0.96	0.65	0.77	1120
6	0.93	0.82	0.87	138
7	0.34	0.58	0.43	26
8	0.76	0.7	0.73	129
9	0.08	0.33	0.13	6
10	0.89	0.64	0.74	270
11	0.32	0.8	0.46	20
12	0.65	0.54	0.59	114
13	0.15	0.4	0.22	15
14	0.12	0.58	0.19	66
15	0.5	0.86	0.63	7
16	0.11	0.5	0.18	2
17	0.73	0.84	0.78	88
18	0.38	0.73	0.5	15
19	0.49	0.57	0.53	44
20	0.37	0.47	0.41	38
21	0.9	0.66	0.76	67
22	0.5	0.5	0.5	2
23	0.35	0.7	0.47	10
24	0.51	0.41	0.45	96
25	0.32	0.5	0.39	46
26	0.57	0.53	0.55	70
27	0.57	0.96	0.71	27
accuracy	0.64	0.64	0.64	1
macro avg	0.46	0.58	0.48	2500
weighted avg	0.79	0.64	0.69	2500

Confusion Matrix (Percentages of True Labels)

True label \ Predicted label	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
0	1	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
3	1	0	0	11	0	0	0	0	0	0	0	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	41	0	0	0	0	0	1	0	1	3	0	1	0	0	0	0	1	0	0	0	7	3	0	0
5	0	0	0	16	1	729	1	0	0	3	3	20	2	1	283	1	7	22	2	3	3	2	0	6	3	0	8	4
6	0	0	0	0	0	2	113	0	0	19	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	3	0	0	15	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
8	10	0	0	3	2	0	0	0	90	0	1	0	6	0	0	0	0	0	0	1	1	0	0	0	6	3	6	0
9	0	0	0	0	0	0	4	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	8	0	0	23	3	0	172	0	4	9	0	0	0	0	14	9	3	1	0	5	8	7	0	4
11	0	0	0	0	0	0	0	0	0	0	0	16	2	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
12	1	0	0	5	1	1	0	0	5	0	1	7	61	6	0	1	0	0	0	3	4	1	1	0	1	12	3	0
13	0	0	0	0	1	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	3	0	0	0	1	4	0	0
14	0	0	0	0	0	25	0	0	0	0	0	0	0	0	38	1	1	1	0	0	0	0	0	0	0	0	0	0
15	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	3	2	0	0	0	1	1	0	0	3	0	0	74	0	1	0	0	0	0	1	0	0	2
18	0	0	0	1	0	0	0	0	0	0	2	0	0	0	0	0	0	0	11	0	0	0	0	0	0	0	1	0
19	0	0	0	3	2	1	1	2	0	1	0	0	1	1	0	0	0	0	0	25	0	0	0	1	3	2	0	1
20	0	0	0	0	1	0	0	0	0	0	0	2	3	3	0	0	0	0	0	0	18	0	0	0	2	8	1	0
21	1	0	0	0	1	0	0	1	3	0	0	0	2	1	0	0	0	0	0	3	4	44	0	0	1	3	3	0
22	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
23	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	7	0	0	0	0
24	5	0	0	1	5	0	0	2	13	0	2	1	4	2	0	0	0	0	1	1	7	0	0	1	39	7	5	0
25	1	0	0	2	1	0	0	1	1	0	1	0	2	6	0	1	0	0	0	1	4	0	0	0	1	23	1	0
26	3	0	0	8	1	1	0	0	0	0	1	1	3	1	0	0	0	0	0	3	1	1	0	0	0	0	37	9
27	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	26

Figure 14 - Balanced Random Forest Results (Test Set)



	precision	recall	f1-score	support
0	0.032	0.8	0.062	5
1	0	0	0	2
2	0	0	0	2
3	0.15	0.56	0.24	16
4	0.37	0.64	0.47	59
5	0.91	0.55	0.69	1120
6	1	0.14	0.24	138
7	0.14	0.77	0.24	26
8	0.85	0.085	0.15	129
9	0.047	0.83	0.088	6
10	0	0	0	270
11	0.24	0.95	0.38	20
12	0	0	0	114
13	0.091	0.2	0.12	15
14	0.089	0.65	0.16	66
15	0.44	1	0.61	7
16	0	0	0	2
17	0.68	0.6	0.64	88
18	0.18	0.67	0.28	15
19	0.37	0.52	0.43	44
20	0.17	0.34	0.23	38
21	0.49	0.61	0.54	67
22	0	0	0	2
23	0.1	0.7	0.18	10
24	0.33	0.01	0.02	96
25	0.3	0.54	0.39	46
26	0.37	0.33	0.35	70
27	0.37	0.93	0.53	27
accuracy	0.41	0.41	0.41	0
macro avg	0.28	0.44	0.25	2500
weighted avg	0.61	0.41	0.43	2500

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
0	4	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0
3	1	0	0	9	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	3	0	1	0	0	0	0	0	0
4	3	0	0	0	38	0	0	3	0	0	0	1	0	4	0	0	0	0	1	0	3	0	0	2	0	3	1	0
5	0	0	0	8	1	617	0	0	0	2	0	28	0	1	424	0	0	12	1	2	1	6	0	8	0	0	3	6
6	0	0	0	0	0	5	19	0	0	96	0	0	0	0	1	0	0	11	4	0	0	0	0	0	0	0	0	2
7	1	0	0	0	2	0	0	20	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0
8	73	0	0	7	4	1	0	2	11	0	0	1	0	0	1	2	0	0	1	3	2	6	0	1	0	2	12	0
9	0	0	0	0	0	1	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	4	0	0	2	26	3	0	96	0	0	0	0	0	4	0	0	0	2	35	20	2	2	0	40	2	9	6	17
11	0	0	0	0	0	0	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
12	7	0	0	10	6	2	0	1	0	0	0	19	0	5	0	3	0	0	0	8	25	10	0	0	0	15	3	0
13	0	0	0	0	0	0	0	2	0	0	0	2	0	3	0	0	0	0	1	0	2	0	0	1	0	3	1	0
14	0	0	0	0	0	21	0	0	0	0	0	0	0	0	43	0	0	0	0	0	0	0	0	1	0	0	0	1
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	1	16	0	1	0	1	0	0	0	0	12	0	0	53	0	0	0	0	0	1	0	0	0	3
18	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	1	0	0	2	0
19	1	0	0	5	1	0	0	2	0	2	0	0	0	3	0	0	0	0	0	23	1	0	0	1	0	3	1	1
20	1	0	0	1	4	0	0	0	0	1	0	3	0	1	0	0	0	0	0	0	13	1	0	1	0	11	1	0
21	2	0	0	3	3	0	0	2	0	0	0	3	0	1	0	1	0	0	0	0	6	41	0	0	0	3	2	0
22	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	7	0	0	0	0
24	22	0	0	6	7	1	0	8	1	0	0	1	0	6	0	1	0	0	3	0	13	8	0	1	1	8	7	2
25	1	0	0	1	6	0	0	2	0	0	0	0	0	4	0	0	0	0	0	0	3	3	0	1	0	25	0	0
26	4	0	0	7	3	5	0	2	0	0	0	1	0	0	1	0	0	0	0	2	5	4	0	1	0	1	23	11
27	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25

Table 1 - Final Model Weighted Log Losses (Test Set)

Model	Weighted Log Loss (Test Set)
OVR Logistic Regression Classifier with L2 Regularisation	0.005225
Balanced Random Forest with Sampling	0.008350

Figure 15 - Class Distribution Across Training & Deployment

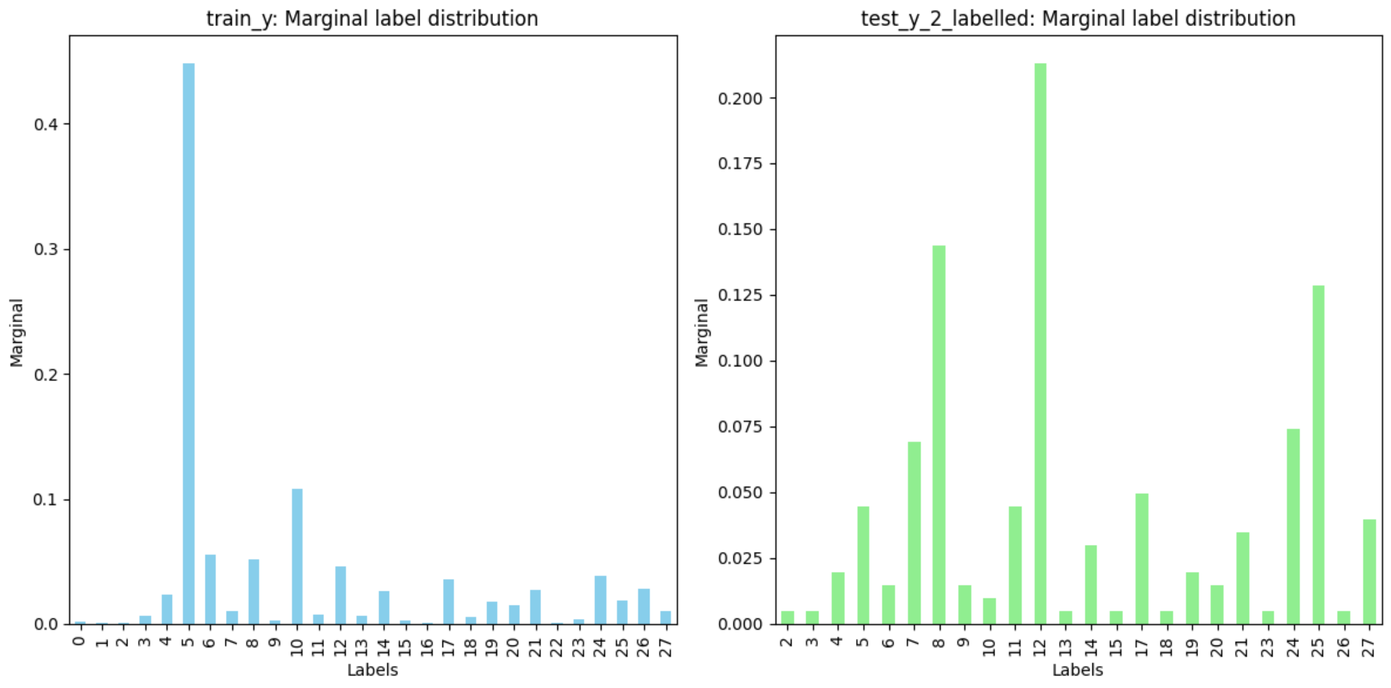


Figure 16 -Marginal class Distribution Across Training & Deployment

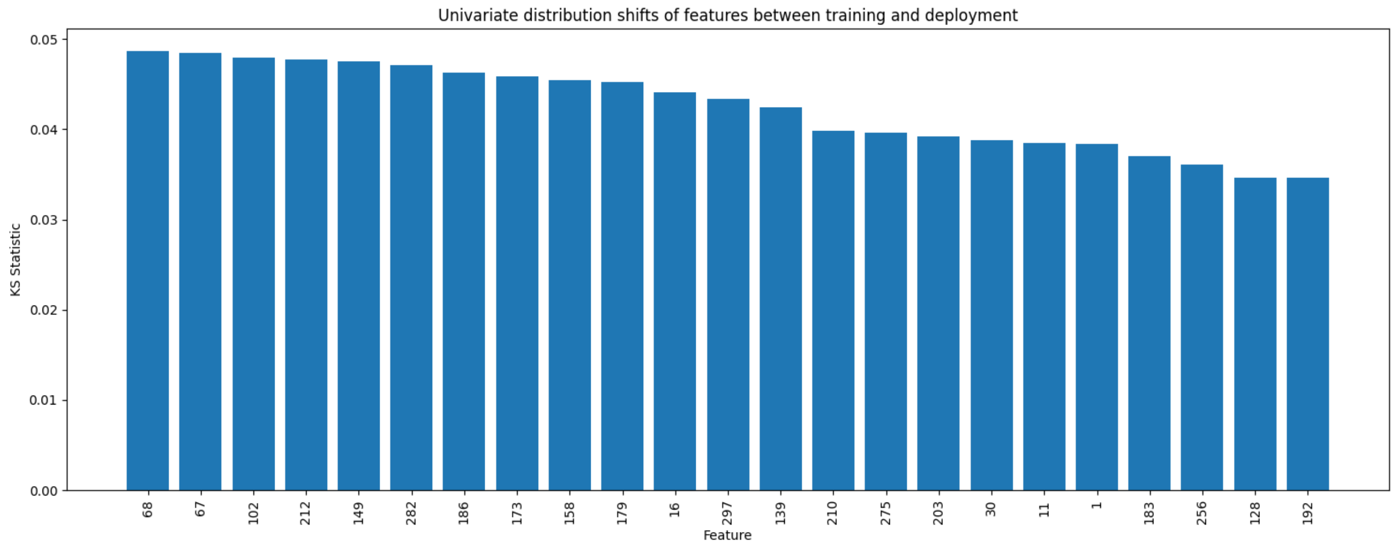


Figure 17 - Histogram of KS Statistics

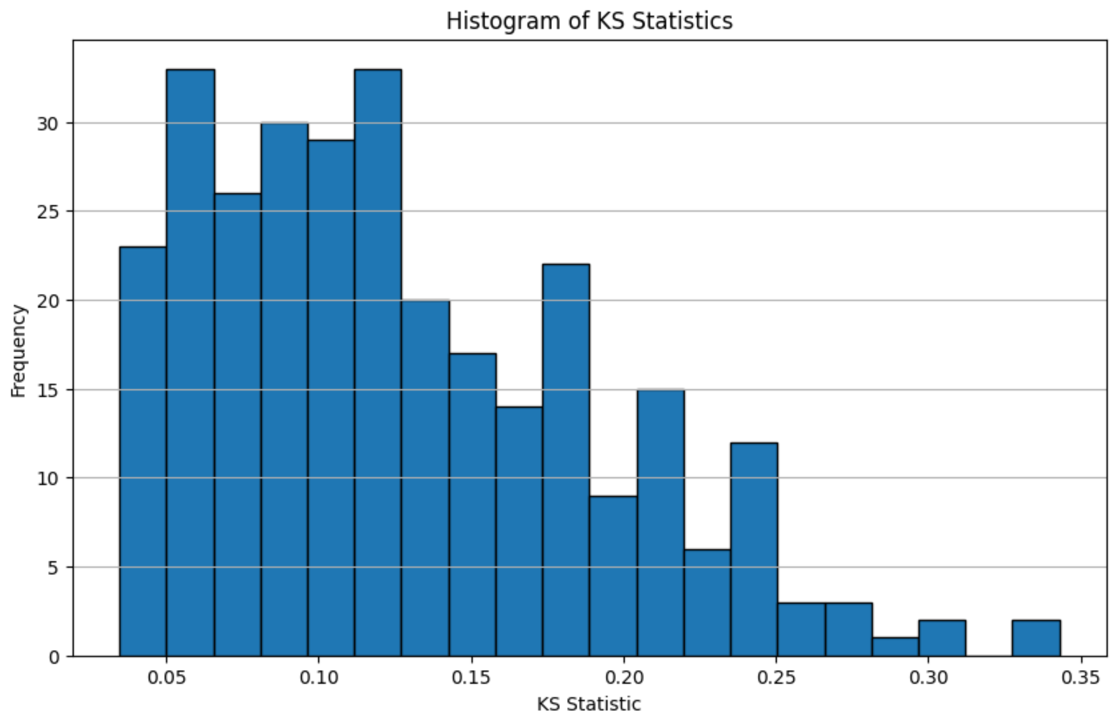
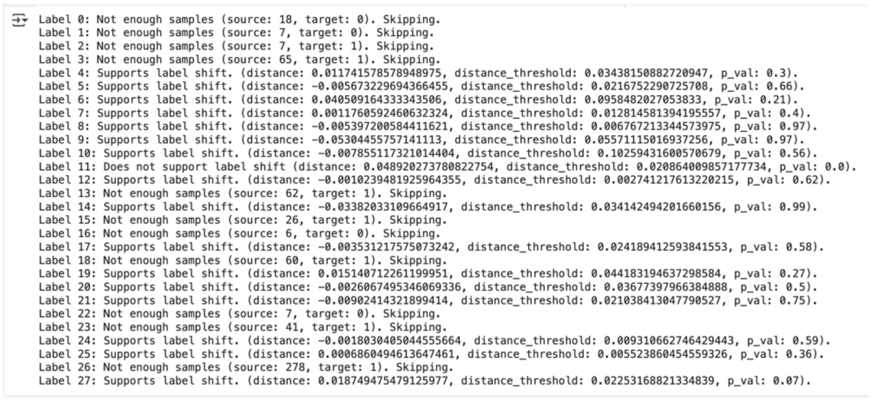
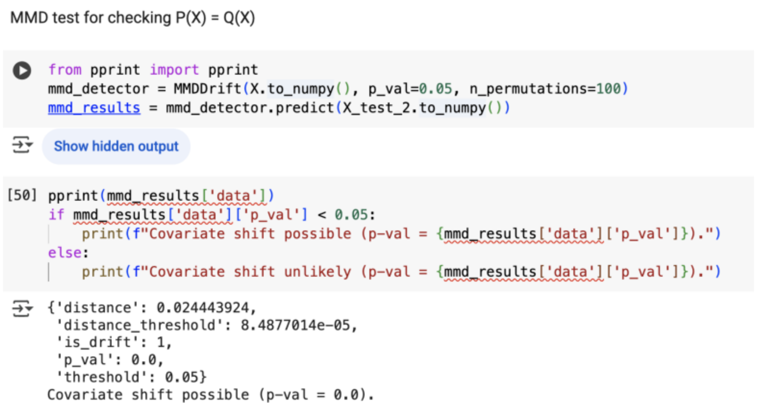


Figure 18 - Maximum Mean Discrepancy Between Training & Deployment ($P(X|y)$ is roughly equivalent to $Q(X|y)$)



```
Label 0: Not enough samples (source: 18, target: 0). Skipping.
Label 1: Not enough samples (source: 7, target: 0). Skipping.
Label 2: Not enough samples (source: 7, target: 1). Skipping.
Label 3: Not enough samples (source: 65, target: 1). Skipping.
Label 4: Supports label shift. (distance: 0.011741578578948975, distance_threshold: 0.03438150882720947, p_val: 0.3).
Label 5: Supports label shift. (distance: -0.005673229694366455, distance_threshold: 0.0216752290725708, p_val: 0.66).
Label 6: Supports label shift. (distance: 0.040509164333343506, distance_threshold: 0.0958482027053833, p_val: 0.21).
Label 7: Supports label shift. (distance: 0.0011760592460632324, distance_threshold: 0.012814581394195557, p_val: 0.4).
Label 8: Supports label shift. (distance: -0.005397200584411621, distance_threshold: 0.006767213344573975, p_val: 0.97).
Label 9: Supports label shift. (distance: -0.05304455757141113, distance_threshold: 0.05571115016937256, p_val: 0.97).
Label 10: Supports label shift. (distance: -0.007855117321014404, distance_threshold: 0.10259431600570679, p_val: 0.56).
Label 11: Does not support label shift (distance: 0.048920273780822754, distance_threshold: 0.020864009857177734, p_val: 0.0).
Label 12: Supports label shift. (distance: -0.0010239481925964355, distance_threshold: 0.002741217613220215, p_val: 0.62).
Label 13: Not enough samples (source: 62, target: 1). Skipping.
Label 14: Supports label shift. (distance: -0.03382033109664917, distance_threshold: 0.034142494201660156, p_val: 0.99).
Label 15: Not enough samples (source: 26, target: 1). Skipping.
Label 16: Not enough samples (source: 6, target: 0). Skipping.
Label 17: Supports label shift. (distance: -0.003531217575073242, distance_threshold: 0.024189412593841553, p_val: 0.58).
Label 18: Not enough samples (source: 60, target: 1). Skipping.
Label 19: Supports label shift. (distance: 0.015140712261199951, distance_threshold: 0.044183194637298584, p_val: 0.27).
Label 20: Supports label shift. (distance: -0.0026067495346069336, distance_threshold: 0.03677397966384888, p_val: 0.5).
Label 21: Supports label shift. (distance: -0.00902414321899414, distance_threshold: 0.021038413047790527, p_val: 0.75).
Label 22: Not enough samples (source: 7, target: 0). Skipping.
Label 23: Not enough samples (source: 41, target: 1). Skipping.
Label 24: Supports label shift. (distance: -0.0018030405044555664, distance_threshold: 0.009310662746429443, p_val: 0.59).
Label 25: Supports label shift. (distance: 0.0006860494613647461, distance_threshold: 0.005523860454559326, p_val: 0.36).
Label 26: Not enough samples (source: 278, target: 1). Skipping.
Label 27: Supports label shift. (distance: 0.018749475479125977, distance_threshold: 0.02253168821334839, p_val: 0.07).
```

Figure 19 - MDD Test for Checking $P(X) = Q(X)$



MMD test for checking P(X) = Q(X)

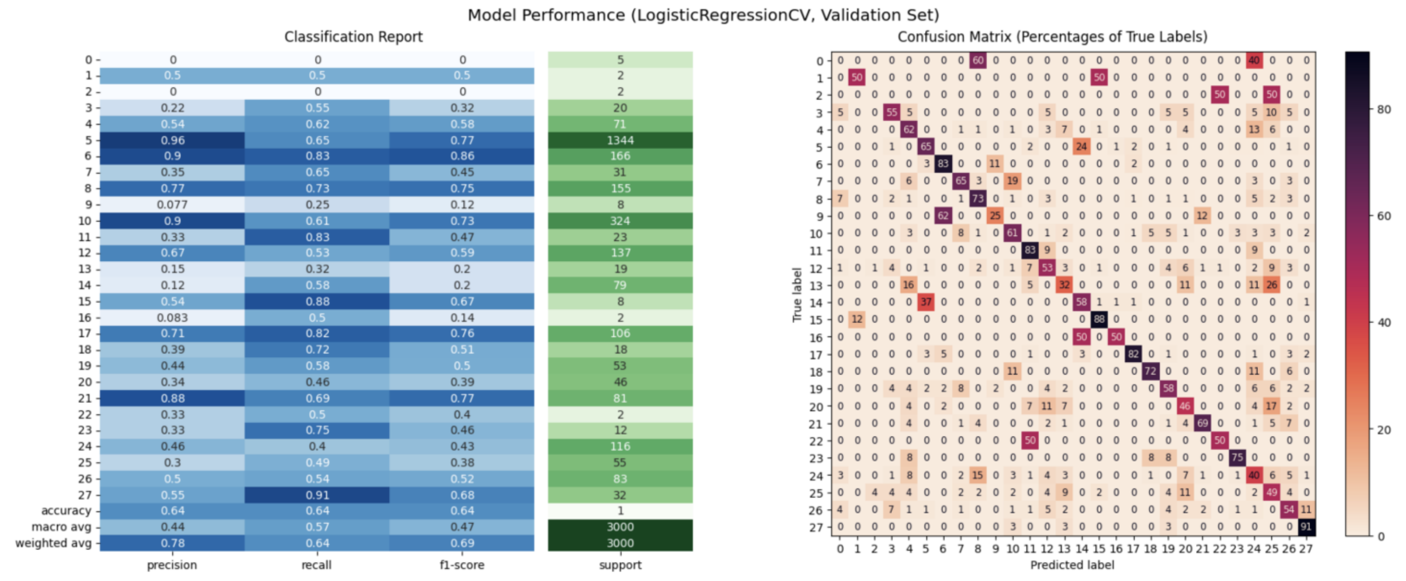
```
from pprint import pprint
mmd_detector = MMDDrift(X.to_numpy(), p_val=0.05, n_permutations=100)
mmd_results = mmd_detector.predict(X_test_2.to_numpy())
```

```
[50] pprint(mmd_results['data'])
     if mmd_results['data']['p_val'] < 0.05:
         print(f"Covariate shift possible (p-val = {mmd_results['data']['p_val']}).")
     else:
         print(f"Covariate shift unlikely (p-val = {mmd_results['data']['p_val']}).")
```

```
{'distance': 0.024443924,
 'distance_threshold': 8.4877014e-05,
 'is_drift': 1,
 'p_val': 0.0,
 'threshold': 0.05}
Covariate shift possible (p-val = 0.0).
```

Figure 20 - Model Performance of Logistic Regression on Source Validation Set

```
Weighted log-loss on validation set: 0.004596770546911202
```



Model Performance (LogisticRegressionCV, Validation Set)

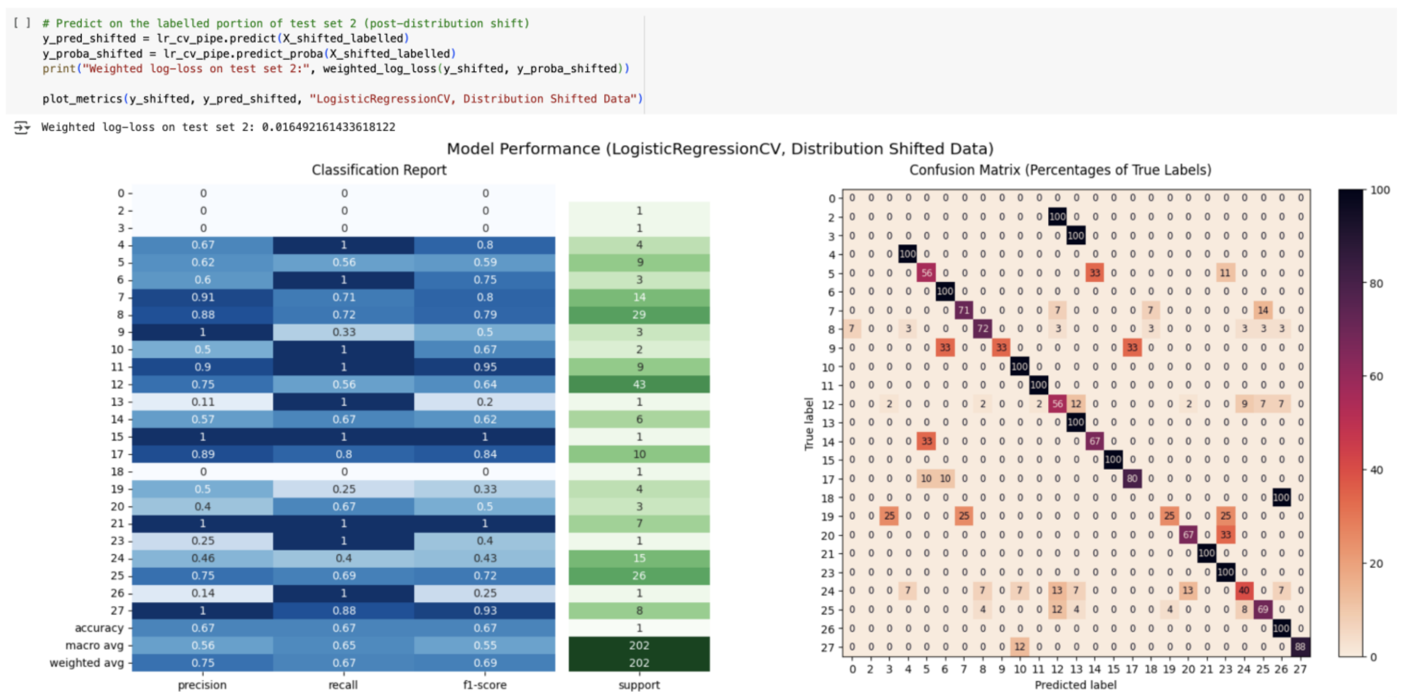
Classification Report

	precision	recall	f1-score	support
0	0	0	0	5
1	0.5	0.5	0.5	2
2	0	0	0	2
3	0.22	0.55	0.32	20
4	0.54	0.62	0.58	71
5	0.96	0.65	0.77	1344
6	0.9	0.83	0.86	166
7	0.35	0.65	0.45	31
8	0.77	0.73	0.75	155
9	0.077	0.25	0.12	8
10	0.9	0.61	0.73	324
11	0.33	0.83	0.47	23
12	0.67	0.53	0.59	137
13	0.15	0.32	0.2	19
14	0.12	0.58	0.2	79
15	0.54	0.88	0.67	8
16	0.083	0.5	0.14	2
17	0.71	0.82	0.76	106
18	0.39	0.72	0.51	18
19	0.44	0.58	0.5	53
20	0.34	0.46	0.39	46
21	0.88	0.69	0.77	81
22	0.33	0.5	0.4	2
23	0.33	0.75	0.46	12
24	0.46	0.4	0.43	116
25	0.3	0.49	0.38	55
26	0.5	0.54	0.52	83
27	0.55	0.91	0.68	32
accuracy	0.64	0.64	0.64	1
macro avg	0.44	0.57	0.47	3000
weighted avg	0.78	0.64	0.69	3000

Confusion Matrix (Percentages of True Labels)

True label \ Predicted label	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40	0	0	0
1	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0	50	0	0
3	5	0	0	55	5	0	0	0	0	0	0	0	5	0	0	0	0	0	0	5	5	0	0	0	5	10	5	0
4	0	0	0	0	62	0	0	1	1	0	1	0	3	7	0	1	0	0	0	0	4	0	0	0	13	6	0	0
5	0	0	0	1	0	65	0	0	0	0	0	2	0	0	24	0	1	2	0	1	0	0	0	0	0	0	1	0
6	0	0	0	0	0	3	83	0	0	11	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
7	0	0	0	0	6	0	0	65	3	0	19	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	3	0
8	7	0	0	2	1	0	0	1	73	0	1	0	3	0	0	0	0	1	0	1	1	0	0	0	5	2	3	0
9	0	0	0	0	0	0	62	0	0	25	0	0	0	0	0	0	0	0	0	0	0	12	0	0	0	0	0	0
10	0	0	0	0	3	0	0	8	1	0	61	0	1	2	0	0	0	1	5	5	1	0	0	3	3	3	0	2
11	0	0	0	0	0	0	0	0	0	0	0	83	9	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0
12	1	0	1	4	0	1	0	0	2	0	1	7	53	3	0	1	0	0	0	4	6	1	1	0	2	9	3	0
13	0	0	0	0	16	0	0	0	0	0	0	5	0	32	0	0	0	0	0	0	11	0	0	0	11	26	0	0
14	0	0	0	0	0	37	0	0	0	0	0	0	0	0	58	1	1	1	0	0	0	0	0	0	0	0	0	1
15	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	88	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	50	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	3	5	0	0	0	0	1	0	0	3	0	0	82	0	1	0	0	0	0	1	0	3	2
18	0	0	0	0	0	0	0	0	0	0	11	0	0	0	0	0	0	0	72	0	0	0	0	0	11	0	6	0
19	0	0	0	4	4	2	2	8	0	2	0	0	4	2	0	0	0	0	0	58	0	0	0	0	6	6	2	2
20	0	0	0	0	4	0	2	0	0	0	0	7	11	7	0	0	0	0	0	0	46	0	0	0	4	17	2	0
21	0	0	0	0	4	0	0	1	4	0	0	0	2	1	0	0	0	0	0	1	4	69	0	0	1	5	7	0
22	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	0
23	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	8	8	0	0	0	75	0	0	0	0
24	3	0	0	1	8	0	0	2	15	0	3	1	4	3	0	0	0	0	1	0	7	1	0	1	40	6	5	1
25	0	0	4	4	4	0	0	2	2	0	2	0	4	9	0	2	0	0	0	4	11	0	0	0	2	49	4	0
26	4	0	0	7	1	1	0	1	0	0	1	1	5	2	0	0	0	0	0	4	2	2	0	1	1	0	54	11
27	0	0	0	0	0	0	0	0	0	0	3	0	0	3	0	0	0	0	0	3	0	0	0	0	0	0	0	91

Assess performance on test set 2 (without distribution shift correction)



```
[ ] # Predict on the labelled portion of test set 2 (post-distribution shift)
    y_pred_shifted = lr_cv_pipe.predict(X_shifted_labelled)
    y_proba_shifted = lr_cv_pipe.predict_proba(X_shifted_labelled)
    print("Weighted log-loss on test set 2:", weighted_log_loss(y_shifted, y_proba_shifted))

    plot_metrics(y_shifted, y_pred_shifted, "LogisticRegressionCV, Distribution Shifted Data")
```

```
Weighted log-loss on test set 2: 0.016492161433618122
```

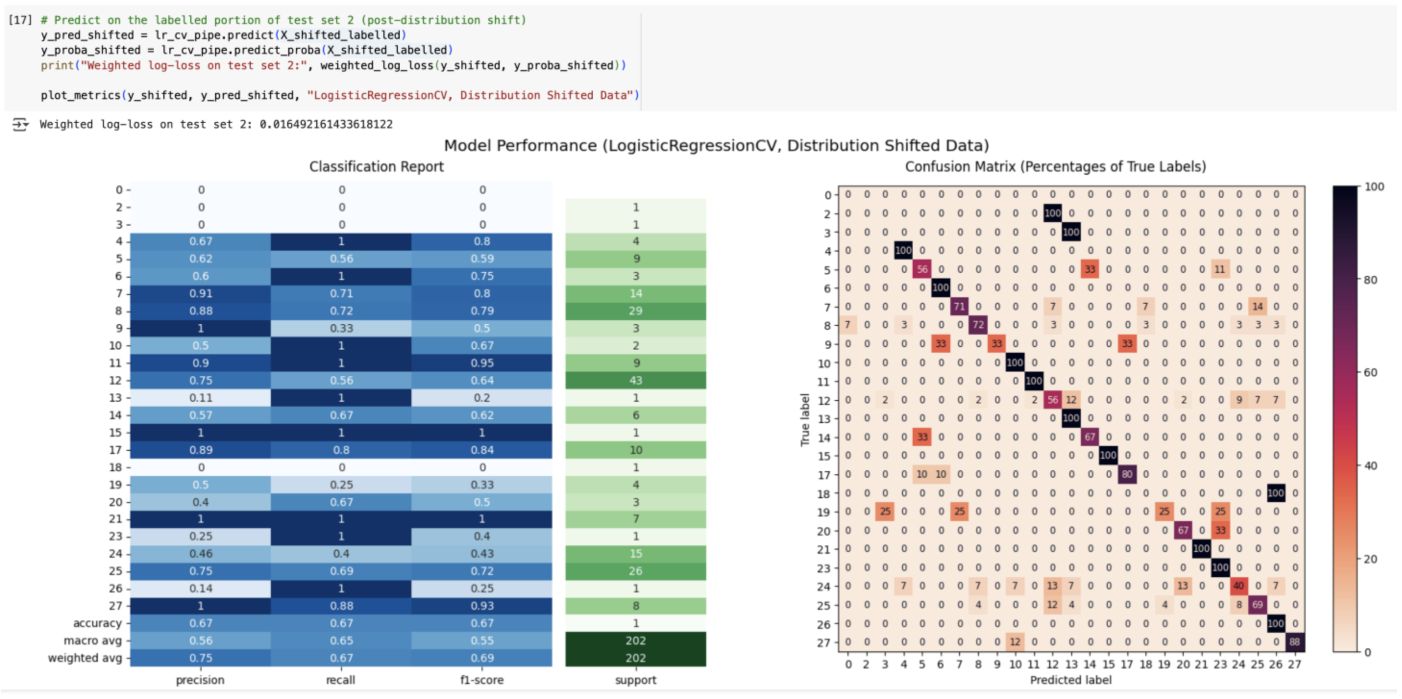
Model Performance (LogisticRegressionCV, Distribution Shifted Data)

Classification Report

	precision	recall	f1-score	support
0	0	0	0	
2	0	0	0	1
3	0	0	0	1
4	0.67	1	0.8	4
5	0.62	0.56	0.59	9
6	0.6	1	0.75	3
7	0.91	0.71	0.8	14
8	0.88	0.72	0.79	29
9	1	0.33	0.5	3
10	0.5	1	0.67	2
11	0.9	1	0.95	9
12	0.75	0.56	0.64	43
13	0.11	1	0.2	1
14	0.57	0.67	0.62	6
15	1	1	1	1
17	0.89	0.8	0.84	10
18	0	0	0	1
19	0.5	0.25	0.33	4
20	0.4	0.67	0.5	3
21	1	1	1	7
23	0.25	1	0.4	1
24	0.46	0.4	0.43	15
25	0.75	0.69	0.72	26
26	0.14	1	0.25	1
27	1	0.88	0.93	8
accuracy	0.67	0.67	0.67	1
macro avg	0.56	0.65	0.55	202
weighted avg	0.75	0.67	0.69	202

Figure 21 - Model Performance on Distribution Shift Test Set

Assess performance on test set 2 (without distribution shift correction)



```
[17] # Predict on the labelled portion of test set 2 (post-distribution shift)
     y_pred_shifted = lr_cv_pipe.predict(X_shifted_labelled)
     y_proba_shifted = lr_cv_pipe.predict_proba(X_shifted_labelled)
     print("Weighted log-loss on test set 2:", weighted_log_loss(y_shifted, y_proba_shifted))

     plot_metrics(y_shifted, y_pred_shifted, "LogisticRegressionCV, Distribution Shifted Data")
```

```
Weighted log-loss on test set 2: 0.016492161433618122
```

Model Performance (LogisticRegressionCV, Distribution Shifted Data)

Classification Report

	precision	recall	f1-score	support
0	0	0	0	
2	0	0	0	1
3	0	0	0	1
4	0.67	1	0.8	4
5	0.62	0.56	0.59	9
6	0.6	1	0.75	3
7	0.91	0.71	0.8	14
8	0.88	0.72	0.79	29
9	1	0.33	0.5	3
10	0.5	1	0.67	2
11	0.9	1	0.95	9
12	0.75	0.56	0.64	43
13	0.11	1	0.2	1
14	0.57	0.67	0.62	6
15	1	1	1	1
17	0.89	0.8	0.84	10
18	0	0	0	1
19	0.5	0.25	0.33	4
20	0.4	0.67	0.5	3
21	1	1	1	7
23	0.25	1	0.4	1
24	0.46	0.4	0.43	15
25	0.75	0.69	0.72	26
26	0.14	1	0.25	1
27	1	0.88	0.93	8
accuracy	0.67	0.67	0.67	1
macro avg	0.56	0.65	0.55	202
weighted avg	0.75	0.67	0.69	202

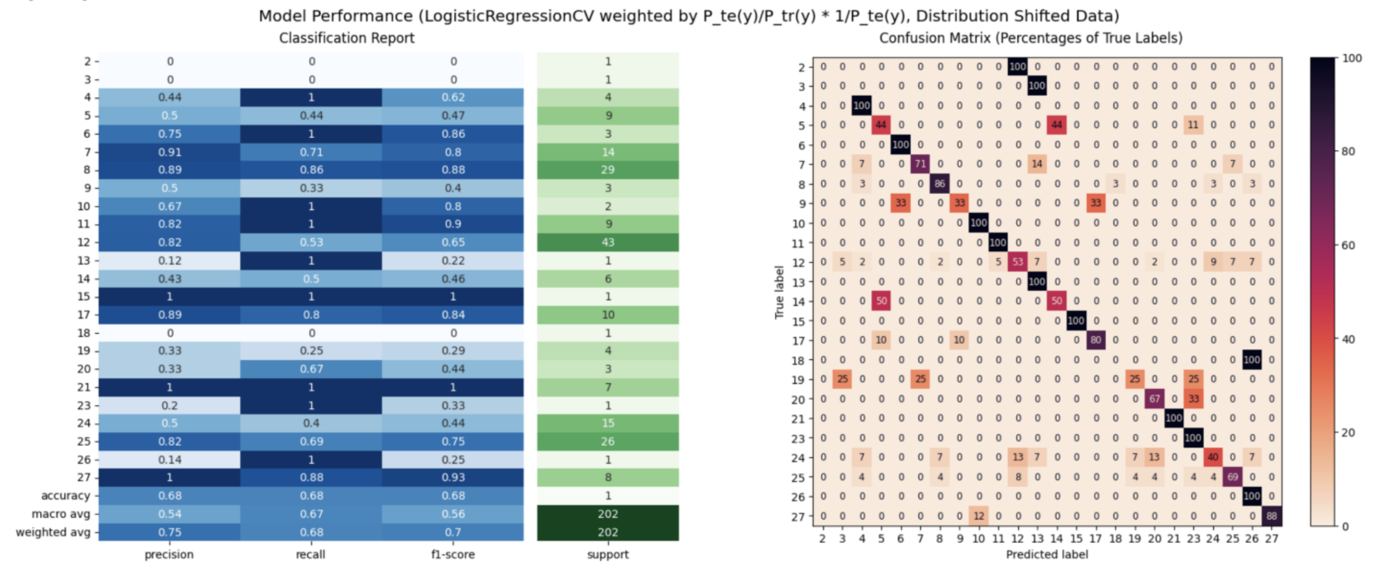
Figure 22 - Model Performance after Correction Label Shift

```
[ ] wt = lshift_sample_weight2 * balanced_shifted_sample_weight
    pipe = Pipeline(steps=[
        ("scaler", scaler),
        ("classifier", lr_label_shift),
    ])

    pipe.fit(X, y, sample_weight=wt)
    print("Best C:", np.median(pipe.named_steps['classifier'].C_))
```

```
Best C: 16.68100537200059
```

```
[ ] # Predict on the target (shifted) distribution
    pred = pipe.predict(X_shifted_labelled)
    proba = pipe.predict_proba(X_shifted_labelled)

    print("Weighted log-loss on shifted dataset:", weighted_log_loss(y_shifted, proba))
    plot_metrics(y_shifted, pred, "LogisticRegressionCV weighted by P_te(y)/P_tr(y) * 1/P_te(y), Distribution Shifted Data")
```

```
Weighted log-loss on shifted dataset: 0.015672572235521848
```



	precision	recall	f1-score	support
2	0	0	0	1
3	0	0	0	1
4	0.44	1	0.62	4
5	0.5	0.44	0.47	9
6	0.75	1	0.86	3
7	0.91	0.71	0.8	14
8	0.89	0.86	0.88	29
9	0.5	0.33	0.4	3
10	0.67	1	0.8	2
11	0.82	1	0.9	9
12	0.82	0.53	0.65	43
13	0.12	1	0.22	1
14	0.43	0.5	0.46	6
15	1	1	1	1
17	0.89	0.8	0.84	10
18	0	0	0	1
19	0.33	0.25	0.29	4
20	0.33	0.67	0.44	3
21	1	1	1	7
23	0.2	1	0.33	1
24	0.5	0.4	0.44	15
25	0.82	0.69	0.75	26
26	0.14	1	0.25	1
27	1	0.88	0.93	8
accuracy	0.68	0.68	0.68	1
macro avg	0.54	0.67	0.56	202
weighted avg	0.75	0.68	0.7	202

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	17	18	19	20	21	23	24	25	26	27
2	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	44	0	0	0	0	0	0	0	0	44	0	0	0	0	0	0	11	0	0	0	0
6	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	7	0	0	71	0	0	0	0	0	14	0	0	0	0	0	0	0	0	0	7	0	0
8	0	0	3	0	0	0	86	0	0	0	0	0	0	0	0	3	0	0	0	0	3	0	3	0
9	0	0	0	0	33	0	0	33	0	0	0	0	0	0	33	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	5	2	0	0	0	2	0	0	5	53	7	0	0	0	0	0	2	0	0	9	7	7	0
13	0	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	50	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0
17	0	0	0	10	0	0	0	10	0	0	0	0	0	0	80	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0
19	0	25	0	0	0	25	0	0	0	0	0	0	0	0	0	0	25	0	0	25	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	67	0	33	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0
24	0	0	7	0	0	0	7	0	0	0	13	7	0	0	0	0	7	13	0	0	40	0	7	0
25	0	0	4	0	0	0	4	0	0	0	8	0	0	0	0	0	4	4	0	4	4	69	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0
27	0	0	0	0	0	0	0	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	88

Appendix 2 – Data Balancing Further Literature Review

Section 2 - SMOTE

SMOTE is an over-sampling technique where "synthetic" samples are generated, rather than duplicating existing ones. These synthetic samples are created along the line segments connecting one or more of the k nearest neighbours from the minority class. Depending on the amount of over-sampling required, neighbours from the k nearest neighbours are chosen at random (Rendón, Alejo, Castorena, Isidro-Ortega, & Granda-Gutiérrez, 2020). At a high level, the algorithm for generating synthetic samples is outlined as follows:

1. Take the difference between the feature vector (sample) under consideration and its nearest neighbour.
2. Multiply the difference by a random number between 0 and 1.
3. Add the difference to the feature vector (sample) under consideration which generates a new synthetic example in feature space.

This results in the selection of a random point along a line segment between the chosen sample and its neighbour. As a result, the decision region of the minority class becomes more general (Chawla, Bowyer, Hall, & Kegelmeyer, 2002). By mitigating bias and ideally capturing important features of the minority class, SMOTE contributes to more accurate prediction and better model performance. However, SMOTE also possesses limitations as it tends to introduce significant noisy data points in the feature space, and it often generates synthetic samples along the same direction, which may complicate the decision boundaries for certain classifiers (Alzubaidi et al., 2023).

Section 3 - ADASYN

ADASYN is another over-sampling technique which generates synthetic samples for minority classes. It is considered as an extension of SMOTE, however, it generates a different number of samples that are dependent on the local distribution of the minority class data. Therefore, more synthetic data is generated for minority class samples that are difficult to learn, compared minority class samples that are easier to learn. The algorithm for ADASYN is as follows:

1. Calculate the ratio of the minority classes to the majority classes to calculate the degree of class imbalance, where $d \in (0, 1]$.

$$d = \frac{m_s}{m_l}$$

Where m_s and m_l is the number of minority class examples and number of majority class examples respectively.

2. If d is less than a certain threshold, then initialise the algorithm.
3. Calculate the total number of synthetic data examples for the minority class:

$$G = (m_l - m_s)\beta$$

Where $\beta \in [0, 1]$ is a parameter which specifies the desired balance level after the synthetic data is generated. A β value of 1 indicated a fully balance dataset after ADASYN.

4. For each minority class x_i, find the k-nearest neighbours based on Euclidean distance in n dimensional space, and calculate r_i:

$$r_i = \frac{\Delta_i}{K}$$

Where $r_i \in [0, 1]$ and Δ_i is the # of examples in the K-nearest-neighbours of x_i in the majority class.

5. Normalise r_i so all the values sum to 1 (a density distribution):

$$\widehat{r_i} = \frac{r_i}{\sum r_i}$$

6. Calculate the number of required synthetic samples for each minority sample x_i given by g_i.

$$g_i = \widehat{r_i} \times G$$

Where G is calculated in step 3 as the total number of synthetic data examples for the minority class.

Important note: As r_i is higher for neighbourhoods that are dominated by majority class samples, in step 6, more minority class examples will be generated, giving ADASYN its adaptive nature.

7. For each x_i, generate g_i synthetic data samples in the following manner.

   a) Loop from 1 to g_i.
   b) Randomly choose one minority data example from K nearest neighbours of x_i, called x_{zi}.
   c) Generate a synthetic data example:

$s_i = x_i + (x_{zi} - x_i)\lambda$

where $(x_{zi} - x_i)$ is the difference vector and $\lambda \in [0, 1]$.

As a result of this algorithm, ADYSYN provides a balanced representation of the data distribution and forces the algorithm to focus on difficult-to-learn examples. This is due to $\widehat{r_i}$, which measures the distribution of weights for different minority classes according to their level of difficulty in learning. This is a key differentiator from SMOTE which only utilises step 7 of the above algorithm to generate equal numbers of synthetic samples for each minority class. As such it is clear ADYSYN's advantages involves improving classification for underrepresented classes, reducing bias towards the majority class and enhancing generalisation ability of machine learning models by focusing on difficult-to-learn minority classes. However, some limitations of ADASYN include increased computational complexity (based on the steps outlined above) to generate synthetic data samples, potential for overfitting as the synthetic samples may not represent the distribution of the minority class and finally sensitivity to noise and outliers in the dataset.

Section 4 - Bagging Under sampling

1.3.2 Bagging

Bagging is a method which involves generating multiple versions of a predictor and then using those to get an aggregated predictor. These multiple versions are formed by a term called bootstrapping, which involves creating a random subset of data by sampling with replacement (Lecture Week 7). According to Leo Breiman, who introduced the bagging algorithm, "tests on real and simulated data sets using classification and regression trees and subset selection … show bagging can give substantial gains in accuracy. The algorithm for bagging is as follows (Week 7 Lecture).

1. Bootstrapping: Create a random subset of data by sampling with replacement. Repeat this k times to generate k subsets.
2. Parallel training: Train these bootstrapped samples independently and in parallel with each other using a weak or base learner.
3. Aggregation: For a classification model take an average out of all the outputs that are predicted by the individual classifiers.

As such, bagging is extremely useful for imbalanced datasets as it directly reduces model variance and combats overfitting effectively. This allows the model to generalise better even when certain classes are underrepresented. By training on diverse bootstrapped samples, if we apply bagging to a collection of low-bias and high-variance models, and we average them, the bias will remain unaffected, while the variance reduces. This is because the aggregated model contains data that unseen by all the training models. This is further valuable with noisy data, or when combined with decision trees or other base learners prone to variance (Week 7 Lecture).

Glossary

- Label shift: The marginal distribution of the label’s changes between training and deployment, while the conditional distribution of the input features given the labels remain the same.
- Covariate shift: The marginal distribution of the inputs changes between training and deployment, while the conditional distribution of the labels given the input features remain the same.
- i.i.d: identically and independently distributed
- Transfer learning uses knowledge from a related domain to improve performance on the target domain. Ng (2016) in his NIPS tutorial highlighted transfer learning as the next driver of machine learning commercial success after supervised learning.
- Adversarial training learns domain-invariant representations that generalise well across different distributions. Domain-Adversarial Neural Networks (DANNs), as proposed by Ganin et al. (2016) in "Domain-Adversarial Training of Neural Networks," specifically use adversarial training to learn domain-invariant features for unsupervised domain adaptation.
- Data Augmentation: Generating synthetic data points that resemble the deployment distribution can help improve the model's ability to generalise. More recent research explores sophisticated augmentation strategies and the impact of different augmentation techniques on model performance (e.g., Cubuk et al., 2019, "AutoAugment: Learning Augmentation Strategies from Data").
- Domain Adaptation: Uses transfer learning or adversarial training to learn a model that performs well on both source (training) and target (deployment) domains, even if their distributions differ.
- Importance Weighting: This technique involves assigning weights to training instances to compensate for the difference between the source and target distributions, increasing model robustness to the changes in the input features in the deployment data.

References

Akila (2016) *(PDF) data imbalance: Effects and solutions for classification of large and highly imbalanced data*. Available at: https://www.researchgate.net/publication/320895020_Data_Imbalance_Effects_and_Solutions_for_Classification_of_Large_and_Highly_Imbalanced_Data (Accessed: 29 April 2025).

BREIMAN, L. (1996) *Bagging predictors*. Available at: https://sci2s.ugr.es/keel/pdf/algorithm/articulo/1996-ML-Breiman-Bagging%20Predictors.pdf (Accessed: 29 April 2025).

Chawla, N.V. *et al.* (no date) *Smote: Synthetic minority over-sampling technique*, *Journal of Artificial Intelligence Research*. Available at: https://www.jair.org/index.php/jair/article/view/10302 (Accessed: 28 April 2025).

Farhadpour, S., Warner, T.A. and Maxwell, A.E. (2024) *Selecting and interpreting multiclass loss and accuracy assessment metrics for classifications with class imbalance: Guidance and best practices*, *MDPI*. Available at: https://www.mdpi.com/2072-4292/16/3/533 (Accessed: 28 April 2025).

He, H. (2013) *Imbalanced learning: Foundations, algorithms, and applications*, *Wiley.com*. Available at: https://www.wiley.com/en-us/Imbalanced+Learning%3A+Foundations%2C+Algorithms%2C+and+Applications-p-9781118074626 (Accessed: 29 April 2025).

He, H. and Garcia, E.A. (2009) *Learning from Imbalanced Data*. Available at: https://www.researchgate.net/publication/224541268_Learning_from_Imbalanced_Data (Accessed: 29 April 2025).

He-ieee (2008) *Adasyn#*, *ADASYN - Version 0.13.0*. Available at: https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.ADASYN.html (Accessed: 29 April 2025).

Kulkarni, A., Chong, D. and Batarseh, F.A. (2021) *Foundations of data imbalance and solutions for a data democracy*, *arXiv.org*. Available at: https://arxiv.org/abs/2108.00071 (Accessed: 28 April 2025).

Lemaître, G. (2016) *(PDF) imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in Machine Learning*. Available at: https://www.researchgate.net/publication/308412408_Imbalanced-learn_A_Python_Toolbox_to_Tackle_the_Curse_of_Imbalanced_Datasets_in_Machine_Learning (Accessed: 29 April 2025).

Padurarlu, C. (2019) *(PDF) dealing with data imbalance in text classification*. Available at: https://www.researchgate.net/publication/336538175_Dealing_with_Data_Imbalance_in_Text_Classification (Accessed: 29 April 2025).

Rendón, E. *et al.* (2020) *Data sampling methods to deal with the big data multi-class imbalance problem*, *MDPI*. Available at: https://www.mdpi.com/2076-3417/10/4/1276 (Accessed: 29 April 2025).

Zhou, Z.-H. and Liu, X.-Y. (2006) *Training cost-sensitive neural networks with methods addressing the class imbalance problem | request PDF*. Available at: https://www.researchgate.net/publication/3297500_Training_cost-sensitive_neural_networks_with_methods_addressing_the_class_imbalance_problem (Accessed: 29 April 2025).

Farhadpour, S., Warner, T.A. and Maxwell, A.E. (2024) *Selecting and interpreting multiclass loss and accuracy assessment metrics for classifications with class imbalance: Guidance and best practices*, *MDPI*. Available at: https://www.mdpi.com/2072-4292/16/3/533 (Accessed: 28 April 2025).

Kulkarni, A., Chong, D. and Batarseh, F.A. (2021) *Foundations of data imbalance and solutions for a data democracy*, *arXiv.org*. Available at: https://arxiv.org/abs/2108.00071 (Accessed: 28 April 2025).

Andrecut, M. (2018) High-Dimensional Vector Semantics, *arXiv.org*. Available at: https://arxiv.org/abs/1802.09914 (Accessed: 28 April 2025).

Mulla, G.A.A., Demir, Y. & Hassan, M.M., 2021. *Combination of PCA with SMOTE oversampling for classification of high-dimensional imbalanced data*. BEÜ Fen Bilimleri Dergisi, 10(3), pp.858–869. https://doi.org/10.17798/bitlisfen.939733.

Cubuk, E.D. et al. (2019) *AutoAugment: Learning Augmentation Strategies from Data*. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Long Beach, CA: IEEE. Available at: https://openaccess.thecvf.com/content_CVPR_2019/papers/Cubuk_AutoAugment_Learning_Augmentation_Strategies_From_Data_CVPR_2019_paper.pdf (Accessed: 29 April 2025).

Ganin, Y. et al. (2016) 'Domain-Adversarial Training of Neural Networks'. *Journal of Machine Learning Research*, 17(59), pp. 1–35. Available at: https://jmlr.org/papers/v17/15-239.html (Accessed: 29 April 2025).

Ng, A. (2016) 'NIPS 2016 Tutorial: Deep Learning'. Tutorial presented at the 30th Conference on Neural Information Processing Systems (NeurIPS), Barcelona, Spain, December 2016. Available at: https://media.neurips.cc/Conferences/NIPS2016/nips2016tutorial-deeplearning.pdf (Accessed: 29 April 2025).

Van Looveren, A. et al. (2021) *Alibi-Detect: Algorithms for Outlier, Adversarial and Drift Detection*. [Python Library]. Version 0.10.0. Available at: https://github.com/SeldonIO/alibi-detect (Accessed: 29 April 2025).

Feigelson, E.D. and Babu, G.J. (2012) *Modern Statistical Methods for Astronomy: With R Applications*. Cambridge: Cambridge University Press. (Accessed: 30 April 2025)