Toggle code

Week 8 - Support Vector Machines

Dr. David Elliott

Notes

- The idea for this workbook/lecture is to focus on demonstrating the models use on different datasets.
  - The Iris dataset is useful for demonstrating SVM's, but are a bit *"too small to be representative of real world machine learning tasks"*[1].
  - On the Iris dataset, we will probably do well no matter how our hyperparameters are set. So for this lecture/workbook I'll demonstrate its use on larger (yet still manageable) datasets.

6. Imballanced Data

Class imbalance is a quite common problem when working with real-world data.

This occours when examples from one class or multiple classes are over-represented in a dataset (e.g. spam filtering, fraud detection, disease screening).

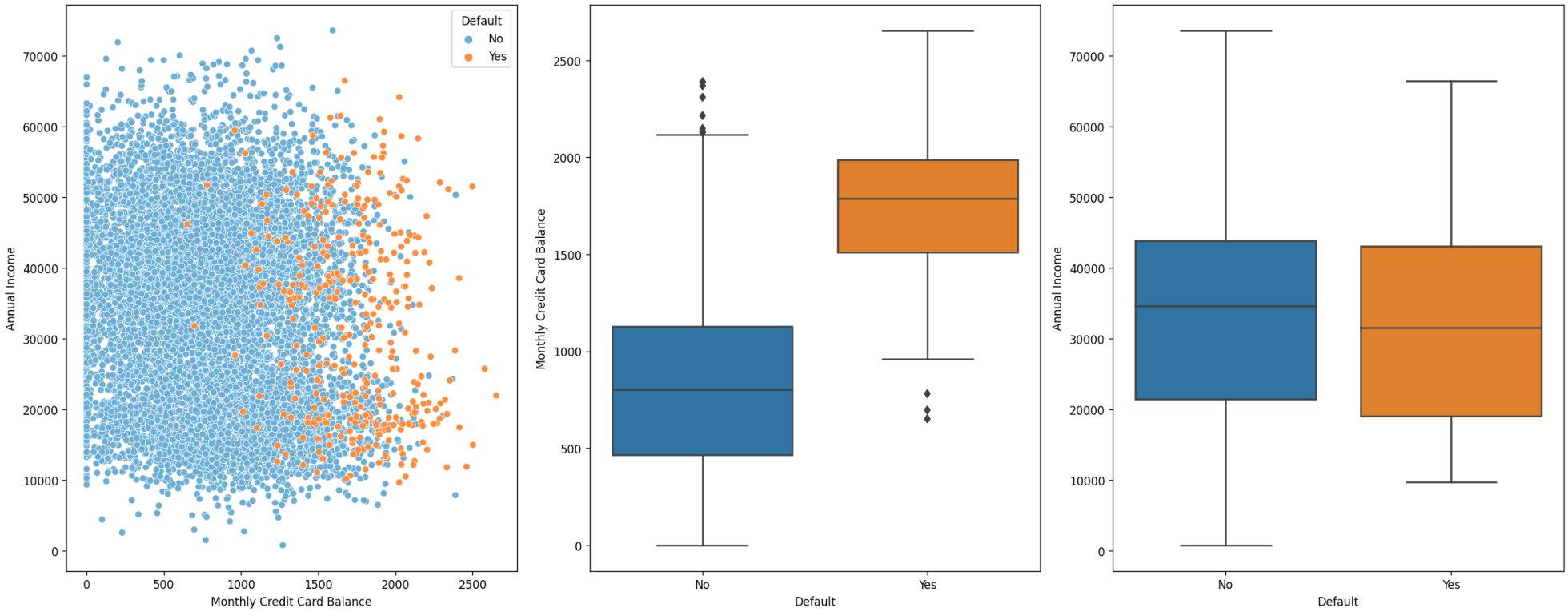
Dataset Example: Default

Default: Customer default records for a credit card company

"We are interested in predicting whether an individual will default on his or her credit card payment, on the basis of annual income and monthly credit card balance."[5]

	default	balance	income
1	No	729.526495	44361.62507
2	No	817.180407	12106.13470
3	No	1073.549164	31767.13895
4	No	529.250605	35704.49394
5	No	785.655883	38463.49588

Figure 30: Annual Income and Monthly Credit Card Balance on Credit Card Payment Defaulting



Notes

- *"We have plotted annual income and monthly credit card balance for a subset of 10, 000 individuals"*[5]
- *"It appears that individuals who defaulted tended to have higher credit card balances than those who did not."*[5]
- Below is a recreation of figure 4.1[5]
- *"It is worth noting that Figure 4.1 displays a very pronounced relationship between the predictor balance and the response default. In most real applications, the relationship between the predictor and the response will not be nearly so strong. However, for the sake of illustrating the classification procedures discussed in this chapter, we use an example in which the relationship between the predictor and the response is somewhat exaggerated."*[5]

```
Class Distribution (%)
No     96.67
Yes     3.33
Name: default, dtype: float64
```

Notes

- The learning and prediction of machine learning algorithms tend to be affected by imballances[6].

Notes

- data is shuffled and stratified in the `train_test_split`

We'll do a random search and we can find a model with high accuracy.

	param_svm_clf__C	mean_test_score	std_test_score
39	13.958286	0.973210	0.003349
44	1.129434	0.972963	0.003252
36	2.1321	0.972963	0.003252
24	17.404635	0.972963	0.003252
22	1.793643	0.972963	0.003252

```
Best Linear Model Accuracy: 96.56%
```

However, this is not much better than a completely useless model that only predicts "No".

```
Useless Model Accuracy: 96.11%
```

Notes

- *"...since only 3.33 % of the individuals in the training sample defaulted, a simple but useless classifier that always predicts that each individual will not default, regardless of his or her credit card balance and student status, will result in an error rate of 3.33 %. In other words, the trivial null classifier will achieve an error rate that null is only a bit higher than the LDA training set error rate."*[5]

This binary classifier can make two types of errors[5]:

- Incorrectly assign an individual **who defaults** to the **no default** category.

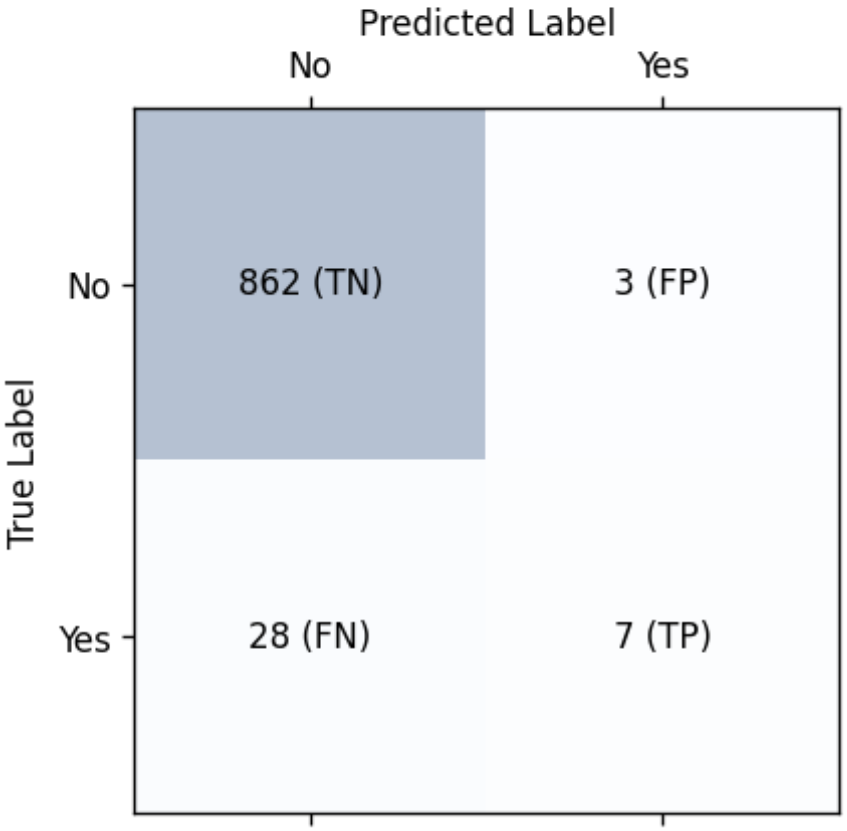
- Incorrectly assign an individual who **does not default** to the **default** category.

While the overall error rate is low, the error rate among individuals who defaulted is very high.

Notes

- From the perspective of a credit card company that is trying to identify high-risk individuals, this error rate among individuals who default may well be unacceptable[5].

Figure 31: Validation Confusion Matrix



Error and Accuracy[1]

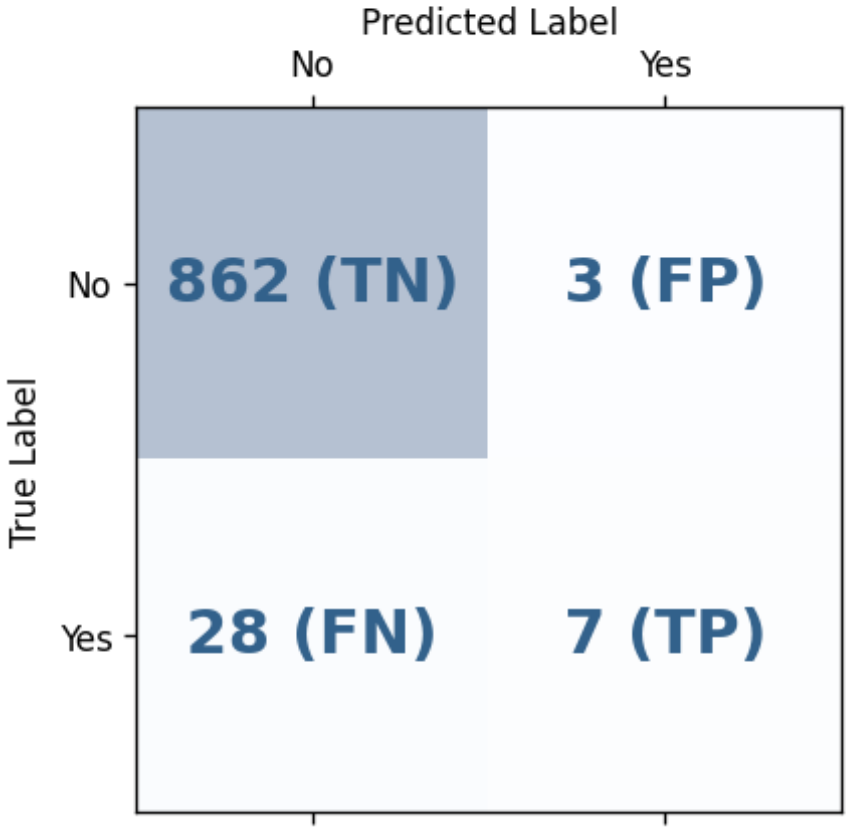
Gives general performance information regarding the number of all correct or false predictions comparative to the total number of predictions for both the positive and negative labels.

$$ERR = \frac{FP + FN}{FP + FN + TP + TN} \tag{1}$$

(2)

$$ACC = 1 - ERR \tag{3}$$

Figure 31: Accuracy Validation Confusion Matrix



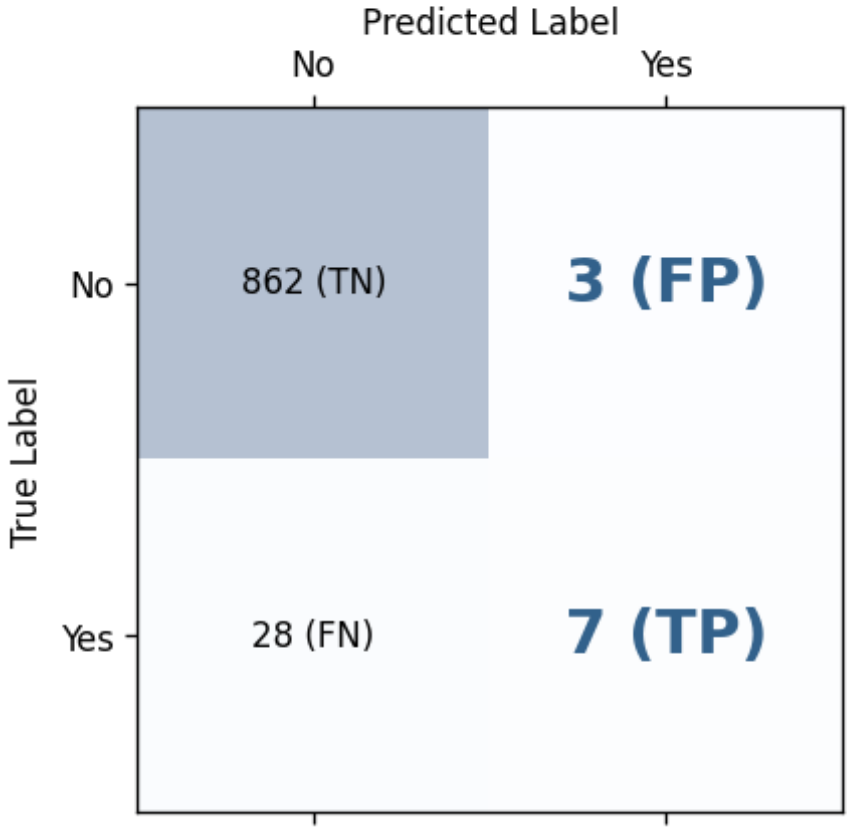
Accuracy (ACC): 0.966

Precision (PRE)[1]

- Precision gives information on how precise your model is by looking at how many positive predicted labels are actually positive.
- Precision is a good measure to determine, when the costs of a False Positive is high.

$$PRE = \frac{TP}{TP + FP}$$

Figure 31: Precision Validation Confusion Matrix



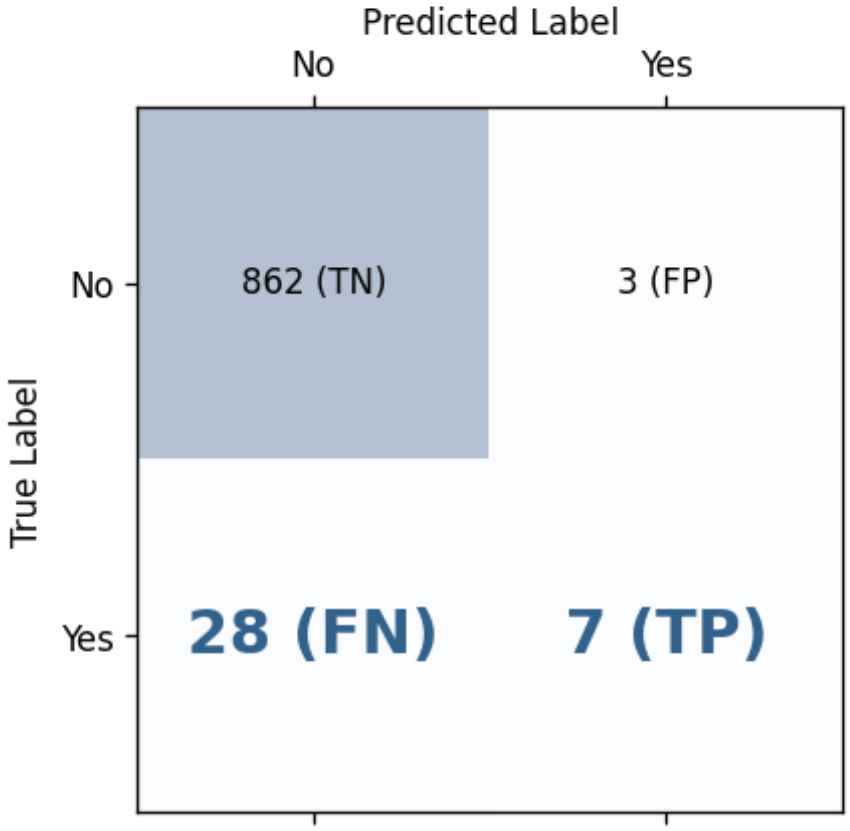
Precision (PRE): 0.700

Recall (or True Positive Rate)[1]

- Calculates how many of the actual positives our model correctly or incorrectly labelled.
- This is useful when the fraction of correctly or misclassified samples in the positive class are of interest.

$$REC = \frac{TP}{FN + TP}$$

Figure 31: Recall Validation Confusion Matrix



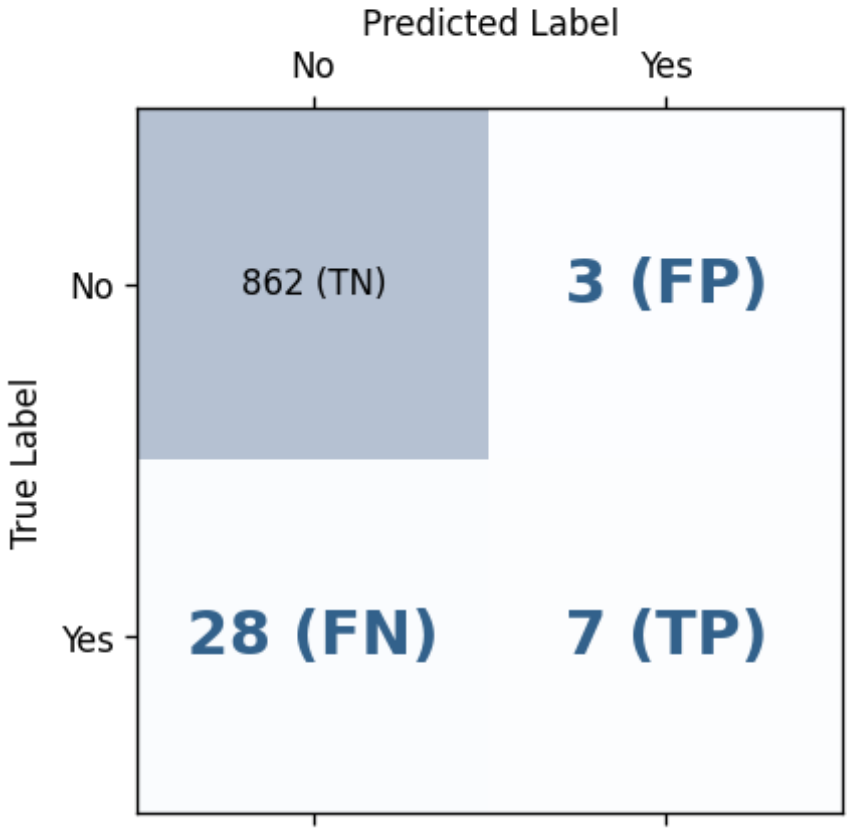
Recall (REC): 0.200

F1-score[1]

- F1-score is a combination of Recall and Precision.
- It is typically used when there is an **uneven class distribution** due to a large number of True Negatives that you are not as focused on.

$$F1 = 2\left(\frac{PRE \times REC}{PRE + REC}\right)$$

Figure 31: F1-Score Validation Confusion Matrix



F1-Score (F1): `0.311`

We can use a classification report, which gives more information such as the macro avg and weighted avg.

Macro Average

- Treats all classes equally as each metric is calculated independently and the average is taken.

Weighted Average

- Each contribution to the average is weighted by the relative number of examples in a class available.

Notes

- The support is the number of occurrences of each class in y_true.
- Notice how the previous metrics were based on the `"yes"` class. This is because in binary classification problems, the default positive label is the target (class 1). You can change this if you are more interested in the other classes performance or the average metrics.

	No	Yes	accuracy	macro avg	weighted avg
precision	0.97	0.70	0.97	0.83	0.96
recall	1.00	0.20	0.97	0.60	0.97
f1-score	0.98	0.31	0.97	0.65	0.96
support	865.00	35.00	0.97	900.00	900.00

Notes

- For further reading on different performance metrics see David M. W. Powers' technical report Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation

Potential Reasons for Poor Performance

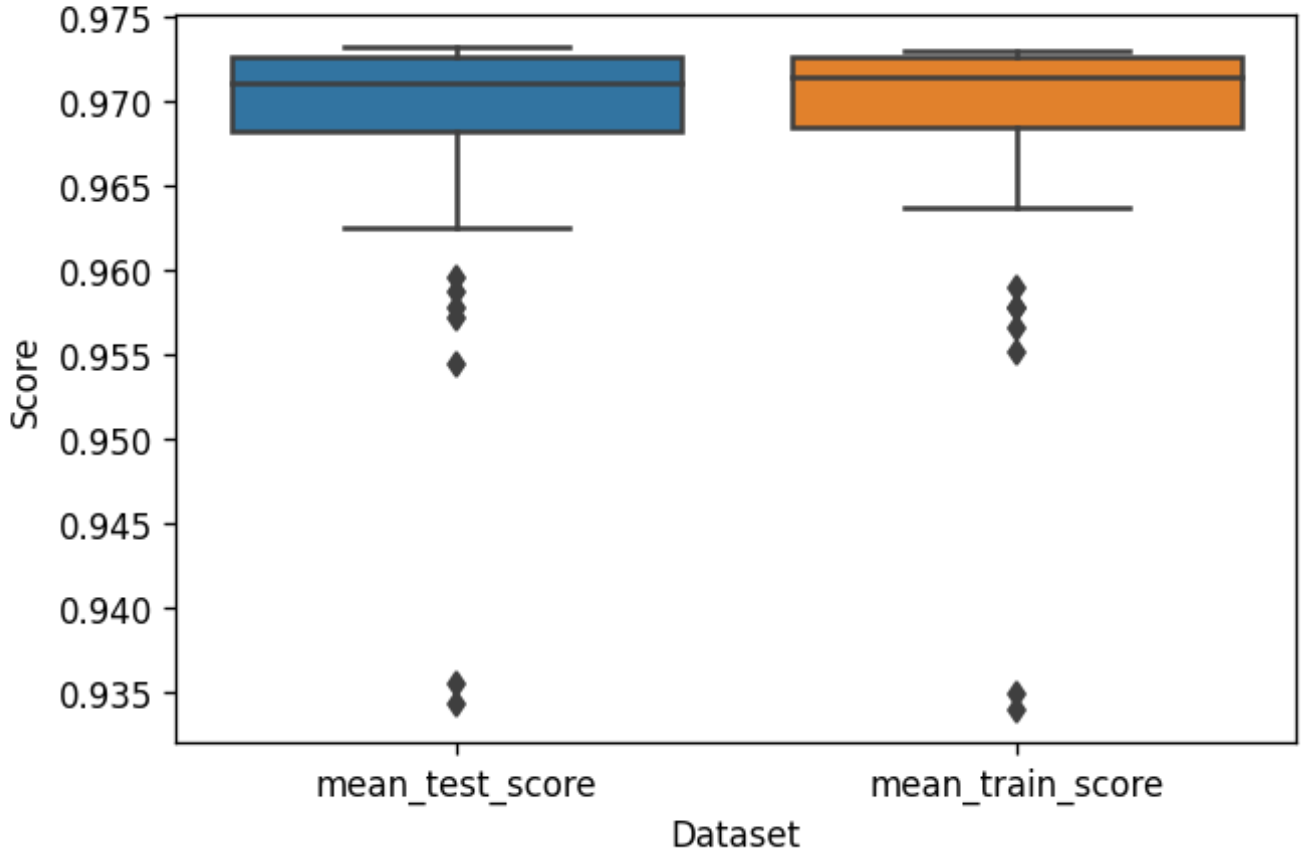
Training Error > Test Error[5]

- Training error rates will usually be lower than test error rates.
- The higher the ratio of parameters p to number of samples n, the more we expect this overfitting to play a role.

Notes

- *"We may expect this classifier to perform worse if we use it to predict whether or not a new set of individuals will default. The reason is that we specifically adjust the parameters of our model to do well on the training data."*[5]
- *"For overfitting these data we don't expect this to be a problem, since p = 2 and n = 10, 000."*

Figure 32: Average Training vs. Validation Accuracy Across Models During GridSearch



Optimising for Accuracy

During hyperparamter cross-validation we are choosing the model with the best **overall accuracy**.

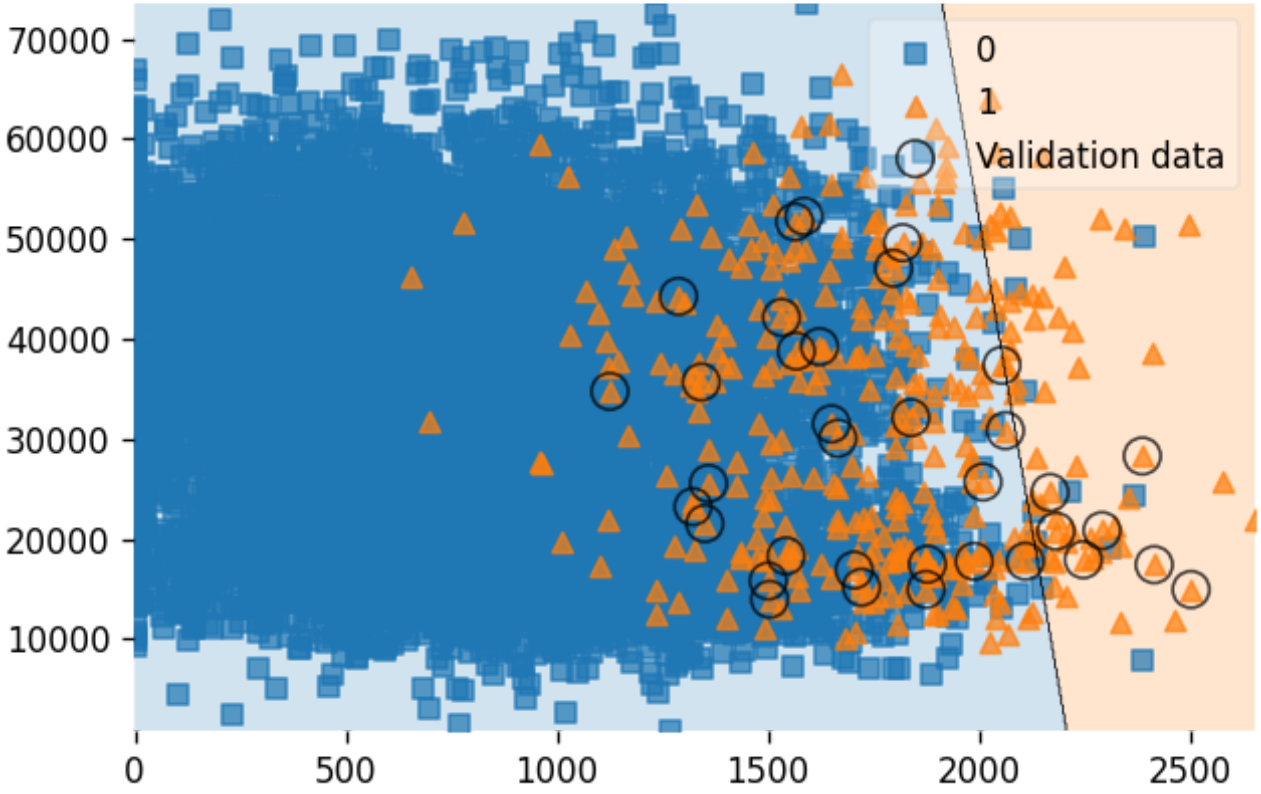
This gives us a model with the smallest possible total number of misclassified observations, irrespective of which class the errors come from[5].

ML algorithms typically optimize a reward or cost function computed as a sum over the training examples, the decision rule is likely going to be biased toward the majority class[9].

Notes

- *"In other words, the algorithm implicitly learns a model that optimizes the predictions based on the most abundant class in the dataset, in order to minimize the cost or maximize the reward during training."*[9].

Figure 33: Model with Best Accuracys Decision Boundary



Potential Solutions for Poor Performance

There are a number of methods available to address imbalances in a dataset, such as:

1. Stratify the k-fold,
2. Weighting the classes in the model during training,
3. Changing the training metric,
4. Resampling the data.

Stratified k-fold

Some of the folds may not have the same amount of data in, so the validation error we get from models may be a poor estimate of performance.

KFold

	Fold 0	Fold 1	Fold 2	Fold 3	Fold 4
0	1565	1572	1577	1570	1560
1	55	48	43	50	60

StratifiedKFold

	Fold 0	Fold 1	Fold 2	Fold 3	Fold 4
0	1568	1569	1569	1569	1569
1	52	51	51	51	51

Weights

During model fitting we can assign a larger penalty to wrong predictions on the minority class.

The heuristic used for `class_weight="balanced"` in Scikit-Learn (0.23.1) is:

$$\frac{n}{Nc \times \sum_{i=1}^{n} I(y_i \in S)},$$

where n are the number of samples, Nc the number of classes, I is an indicator function, and S contains the class elements.

	param_svm_clf__class_weight	param_svm_clf__C	mean_test_accuracy	std_test_accuracy
12	None	12.457135	0.972963	0.002512
19	None	20.185646	0.972963	0.002512
16	None	17.404635	0.972963	0.002512
0	None	5.620905	0.972963	0.002388
52	None	4.498096	0.972963	0.002388

```
C:\Users\delliot2\.conda\envs\mlp_pip\lib\site-packages\sklearn\svm\_base.py:985: ConvergenceWarning: Liblinear failed to conve
rge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

Extra

So far in these notes we have been using a standard classifcation table fom Scikit-Learn, however we may wish instead to use one more suited to imballanced data.

TODO

- explain new metrics

Notes

- In our example case here just changing the weights alone doesn't do much (if anything). We'll see this in a figure later.

	No	Yes	accuracy	macro avg	weighted avg
precision	0.97	0.73	0.97	0.85	0.96
recall	1.00	0.23	0.97	0.61	0.97
f1-score	0.98	0.35	0.97	0.67	0.96
support	865.00	35.00	0.97	900.00	900.00

	No	Yes	avg / total
precision	0.97	0.73	0.96
recall	1.00	0.23	0.97
specificity	0.23	1.00	0.26
f1-score	0.98	0.35	0.96
geo	0.48	0.48	0.48
iba	0.25	0.21	0.24
support	865.00	35.00	900.00

Changing Training/Validation Metric

Changing the metric for what is defined as the *"best model"* can help us prioritise models that make particular errors.

For example, a credit card company might particularly wish to avoid incorrectly classifying an individual who will default, whereas incorrectly classifying an individual who will not default, though still to be avoided, is less problematic.

In this case, **recall** would therefore be a useful metric to use.

Notes

- Now we see that `balanced` models are indeed better if we want a good average recall or f1

	param_svm_clf__class_weight	param_svm_clf__C	mean_test_recall	std_test_recall
9	balanced	0.031589	0.882730	0.021834
10	balanced	0.397242	0.878808	0.029134
37	balanced	2.889491	0.878808	0.029134
1	balanced	0.397409	0.878808	0.029134
14	balanced	0.216116	0.878808	0.029134

	param_svm_clf__class_weight	param_svm_clf__C	mean_test_f1	std_test_f1

	param_svm_clf__class_weight	param_svm_clf__C	mean_test_f1	std_test_f1
32	balanced	61.146821	0.505370	0.035163
41	balanced	55.477953	0.481678	0.035227
23	balanced	76.945463	0.478551	0.054680
18	balanced	38.980718	0.450244	0.038279
36	balanced	124.327323	0.408983	0.033208

```
C:\Users\delliot2\.conda\envs\mlp_pip\lib\site-packages\sklearn\svm\_base.py:985: ConvergenceWarning: Liblinear failed to conve
rge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

Figure 34: Best CV Accuracy Validation Confusion Matrix

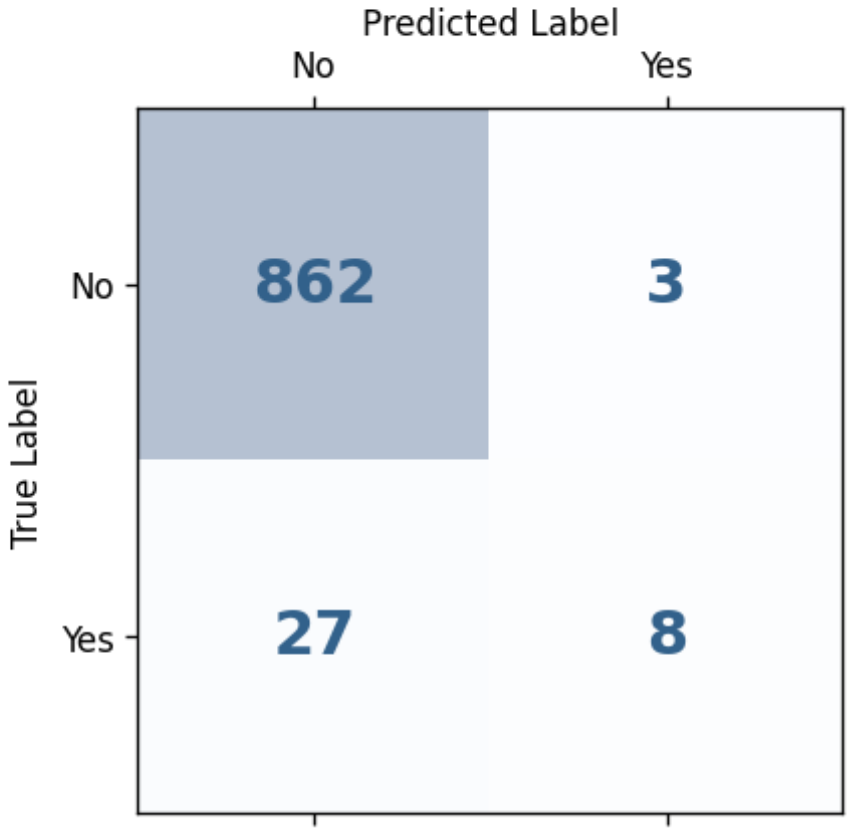
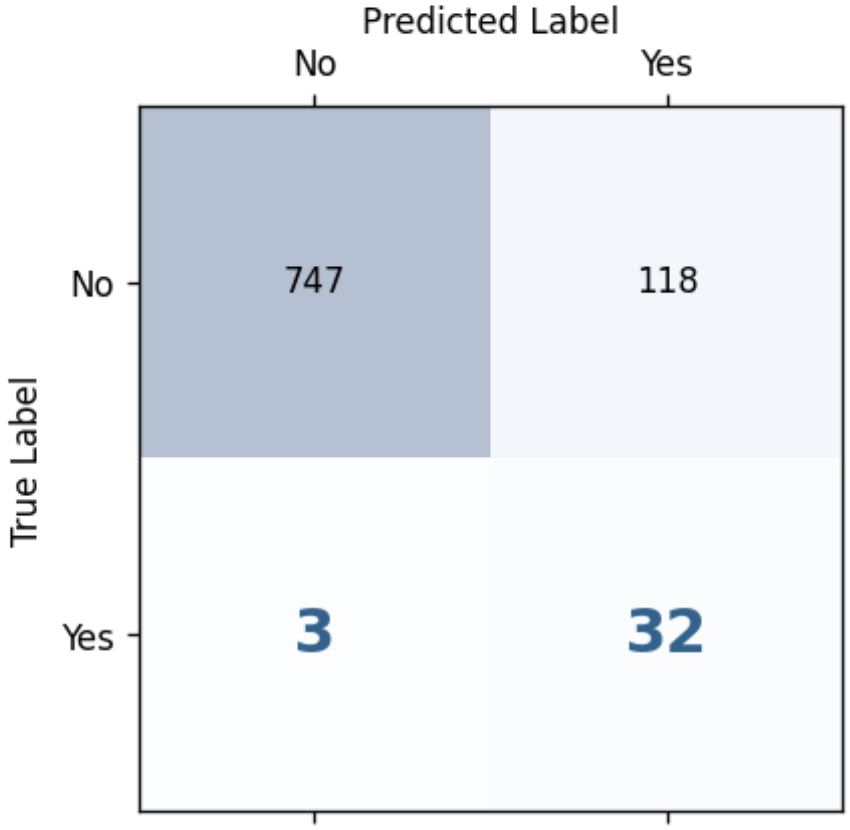


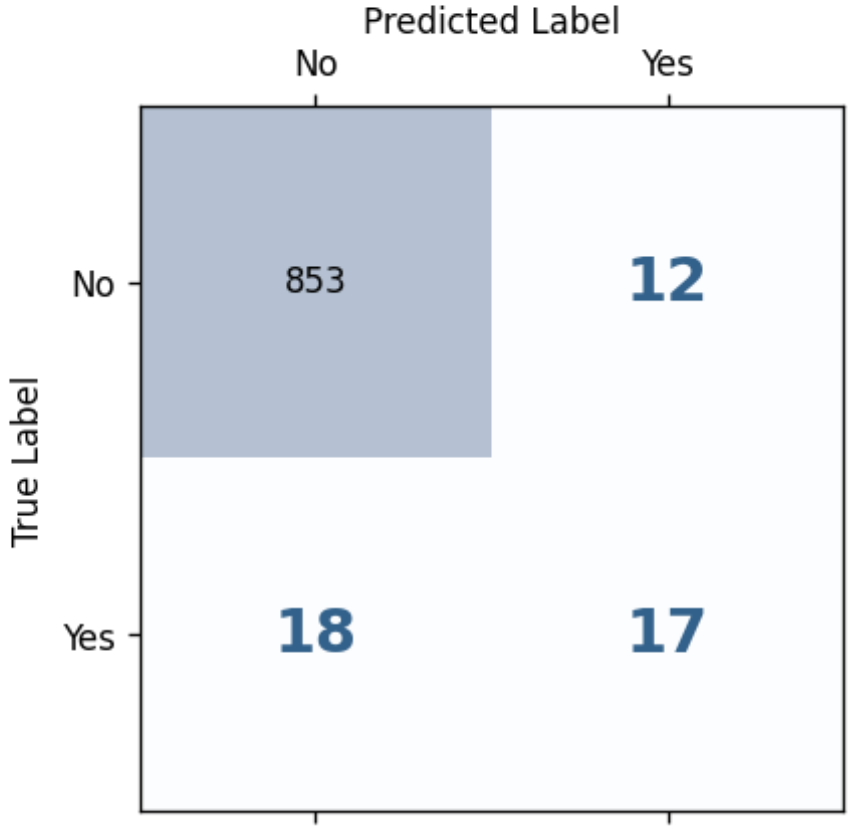
Figure 35: Best CV Recall Validation Confusion Matrix

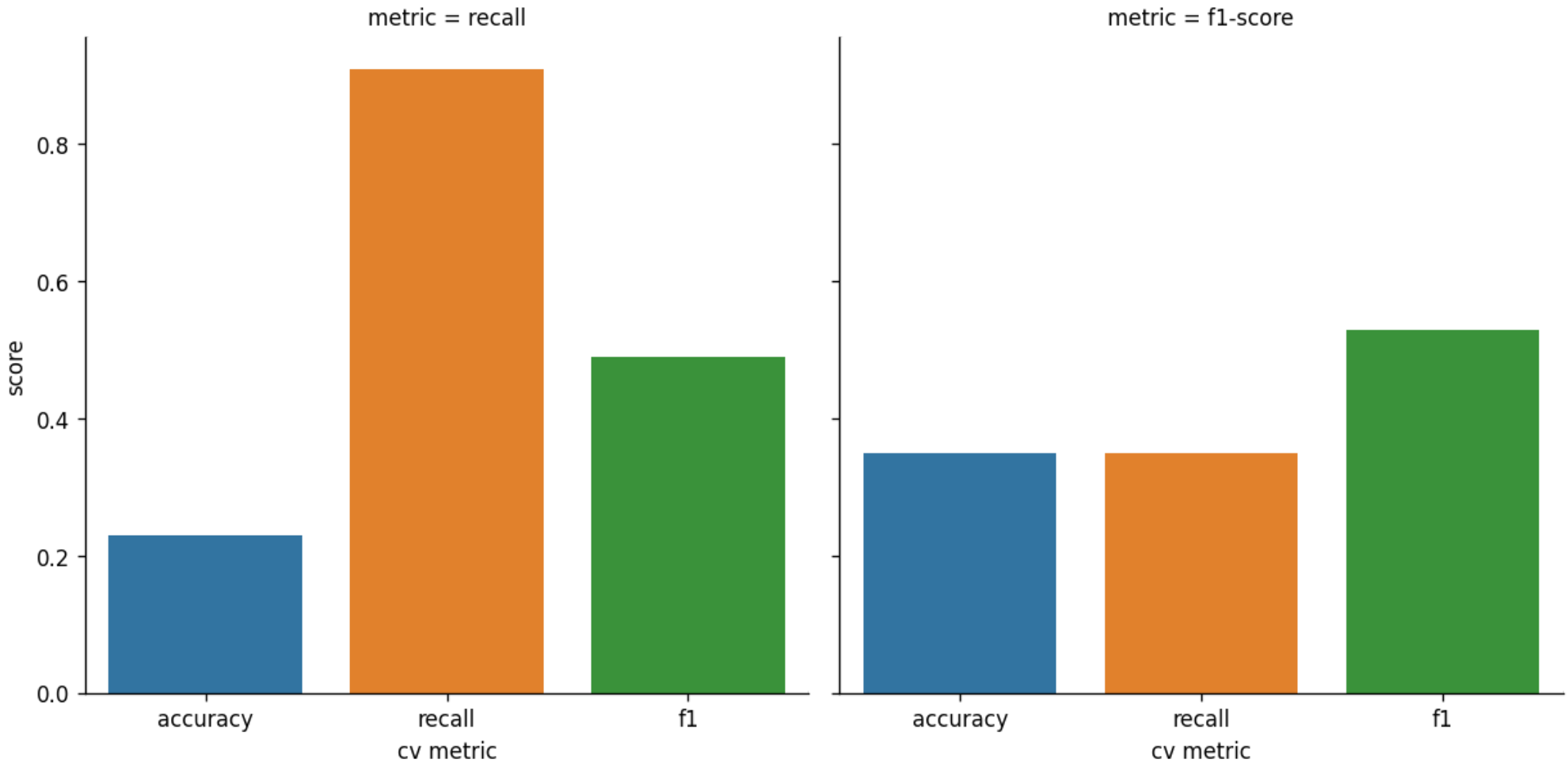


Extra

This is just some additional visualisations.

Figure 36: Best CV F1-Score Validation Confusion Matrix





Providing your comfortable using metrics instead of relying on a `confusion_matrix`, you can use more of your training data by just using the multiple metrics in the `CvGridSearch`.

From here on in, I will get rid of my separate training and validation sets and I will just use `"recall"` as our metric of interest.

Resampling

We can change the distribution of the classes in our training data.

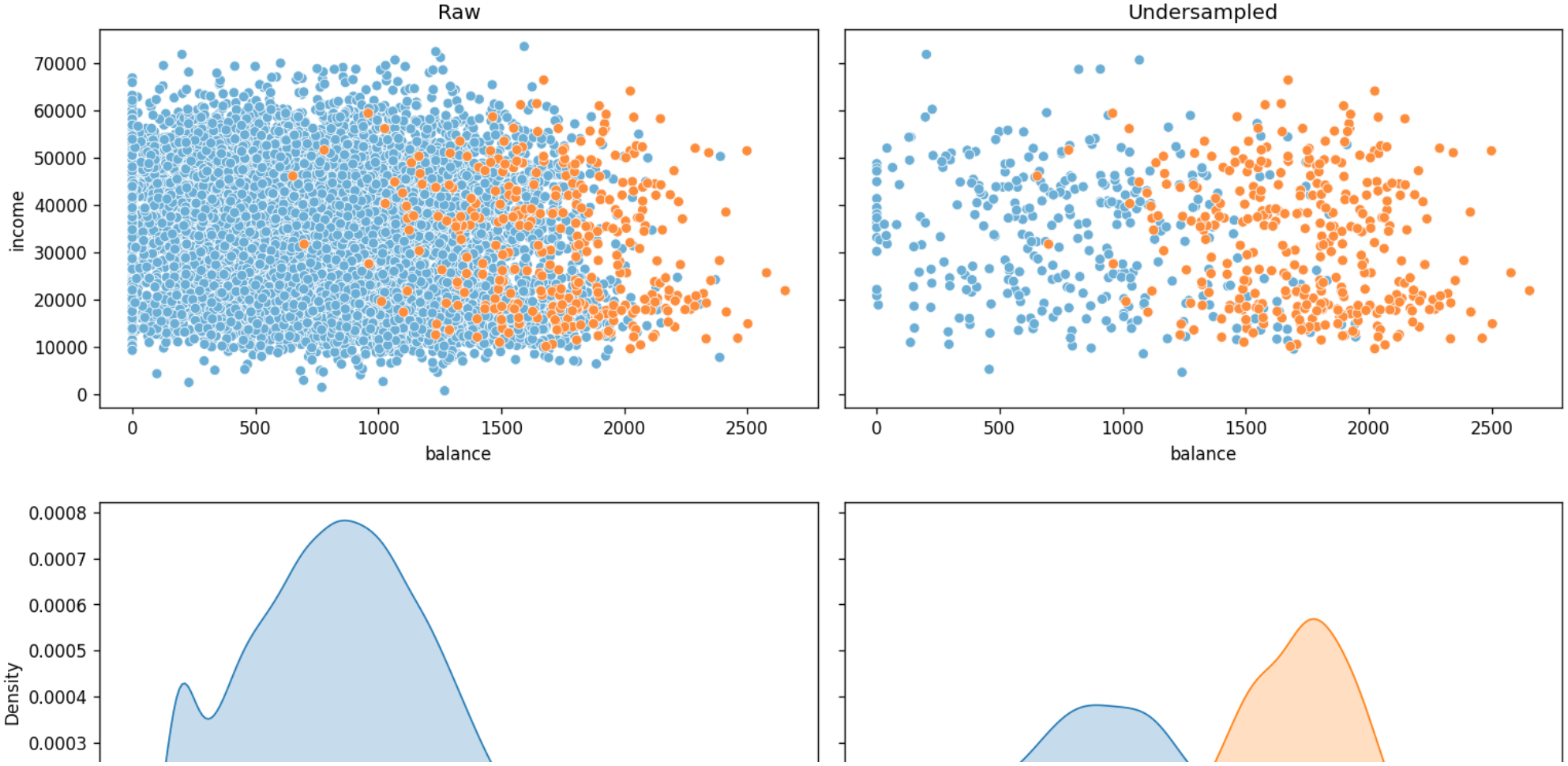
Under-Sampling

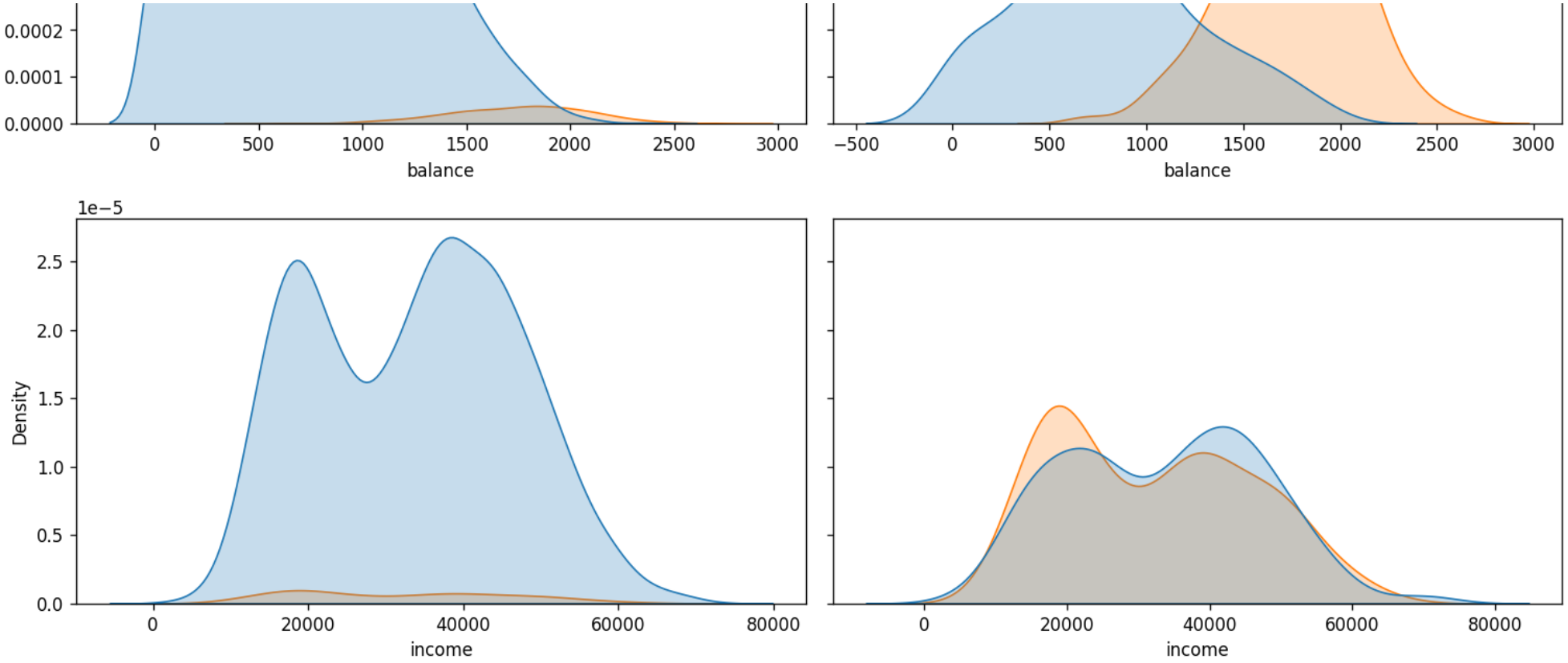
A fast way to balance the data is just to randomly select a subset of the data for each class so they have the number of datapoints found in the smallest class.

Notes

- `RandomUnderSampler` is part of the Imblearn package, which allows for a lot of techniques for working with imballanced data.
- There is a `resample` method in `scikit-learn` but Imblearn is a bit smoother to work with.

Figure 37: RandomUnderSampler





Note

- Make sure your sampler is **in the pipeline**, otherwise you'll be training and testing your data on a smaller/larger sample than normal and get unrepresentative results!
  - this is an easy mistake to make... I did it when setting this up :')

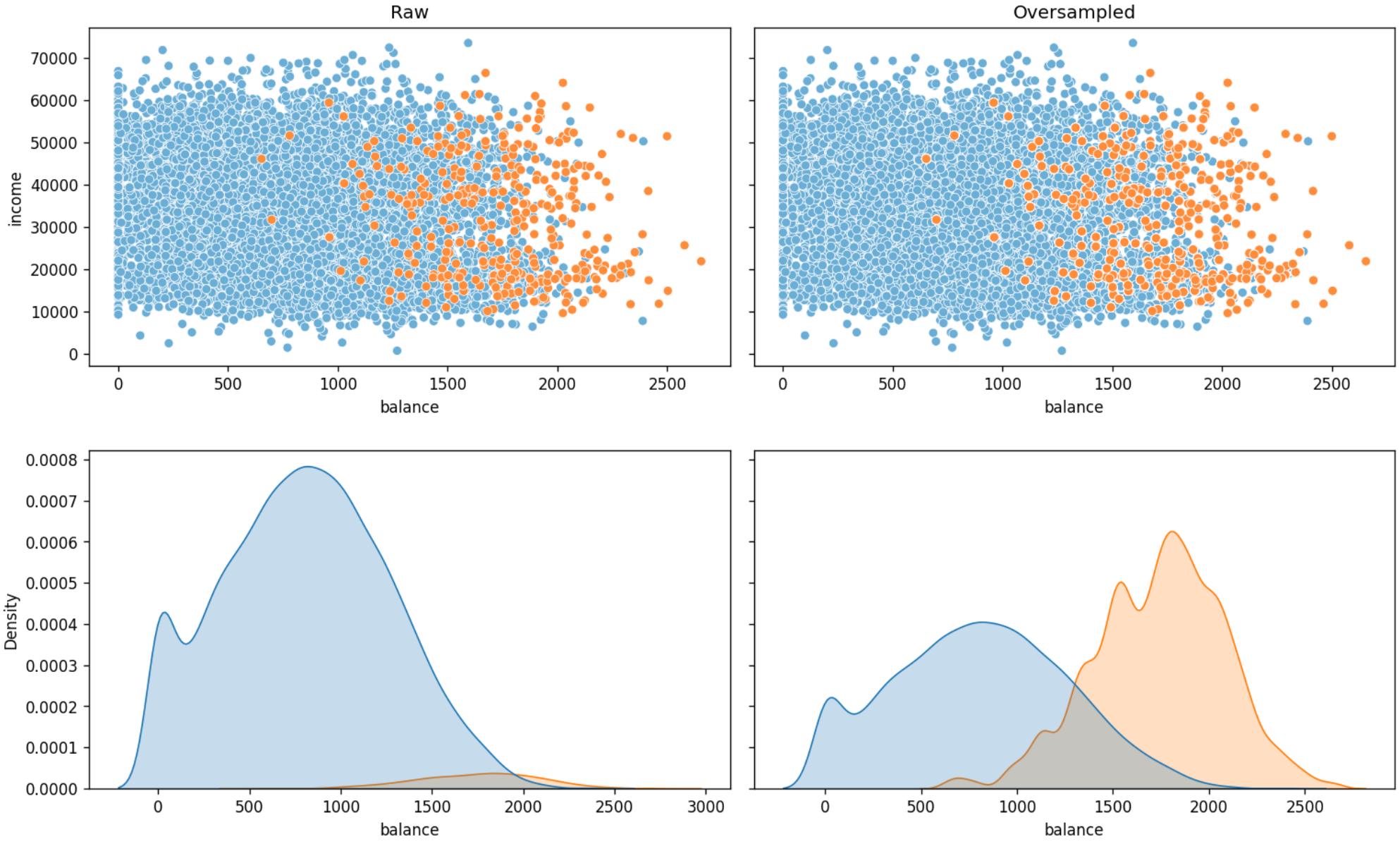
Oversampling

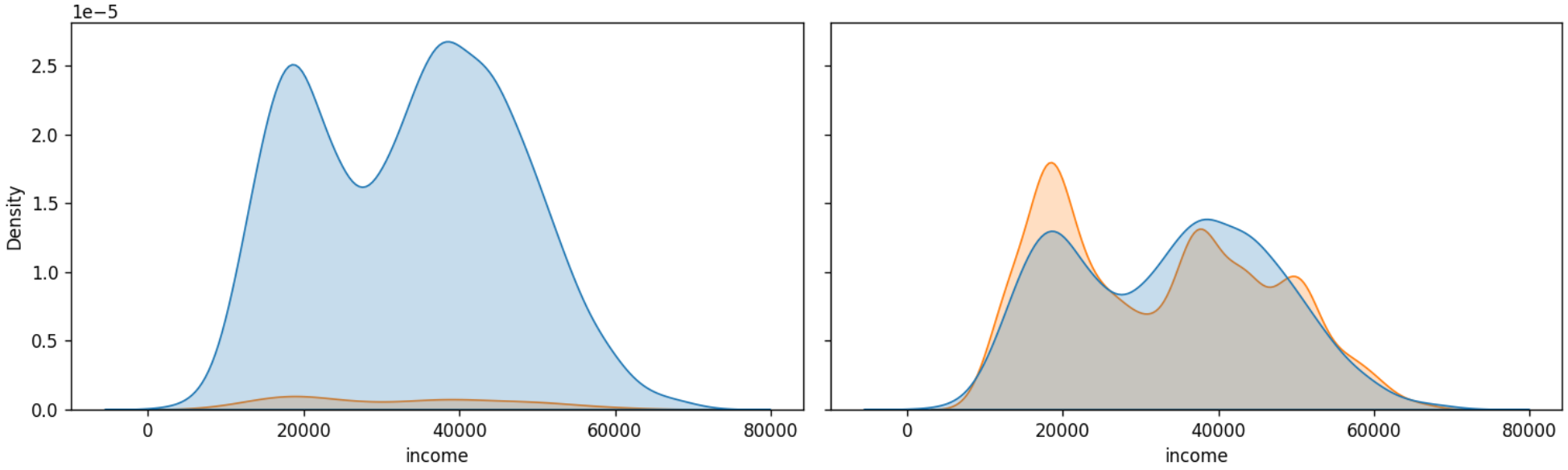
Data can be oversampled easily by randomly sampling from minority classes with replacement to duplicate original samples.

Notes

- make sure to oversample after splitting the training and validation sets or you may "bleed" information into the validation sets of the model when trying to test a model[7]... In-other-words, make sure it is in a pipeline!

Figure 38: RandomOverSampler





We can see if a RBF improves things although, if you plan on running this yourself (`overwrite=True`), this is computationally expensive.

Extra

NearMiss

A number of undersampling methods use heuristics based on k-nearest neighbors (KNN) classification[8]. KNN finds a number of samples that are the most similar to a data point we want to classify, based on a given distance metric, with its assigned class label depending on a majority vote by the nearest neighbours[9] (we'll come back to this later). NearMiss uses this by selecting samples in the class to be under-sampled where the average distance to the closest or farthest samples of the minority class is smallest[10].

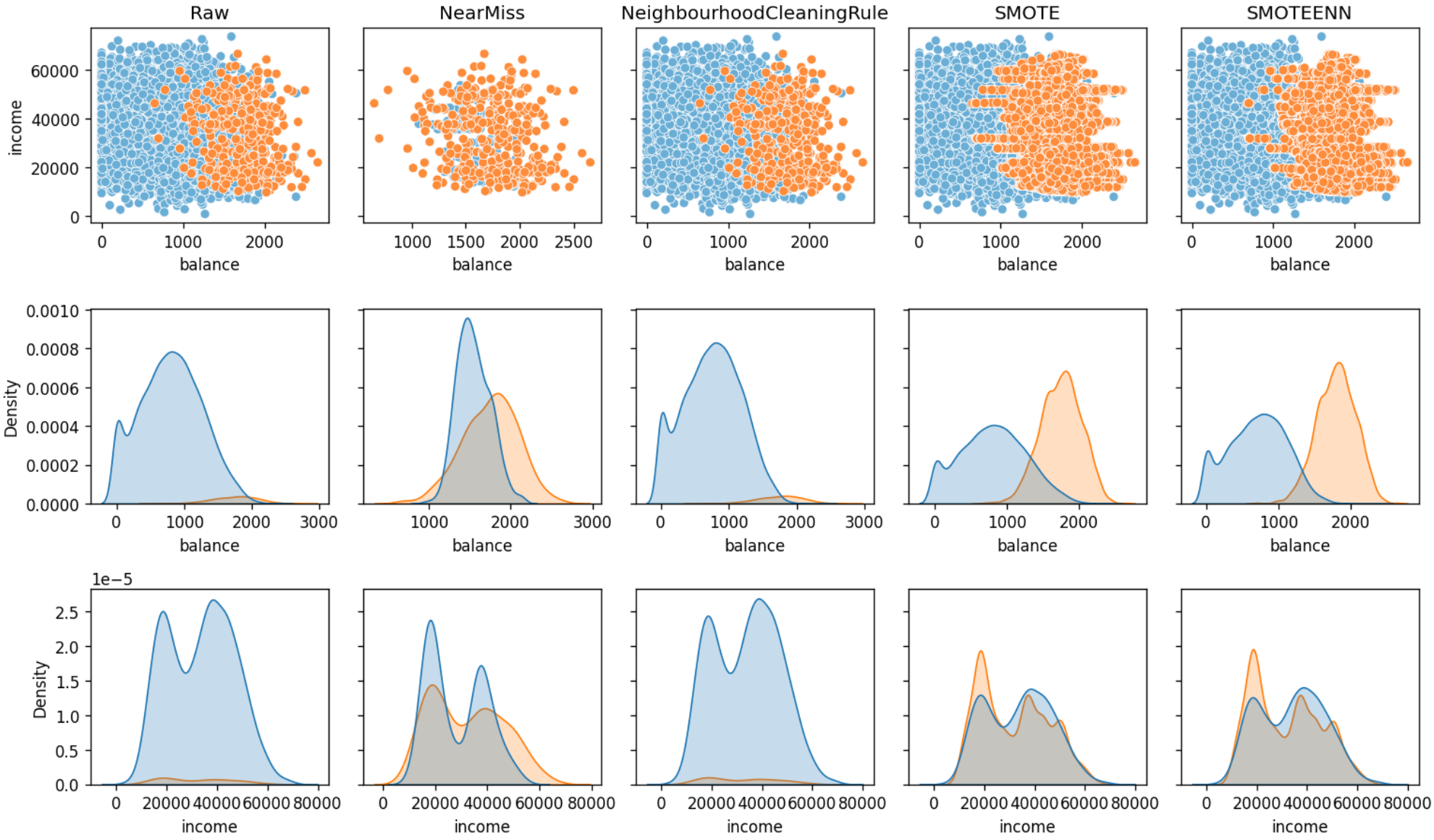
NeighbourhoodCleaningRule

Undersampling techniques also include data cleaning rules, where the number of samples in classes are not specified, but data is edited based on methods such as removing data dissimilar to their neighbourhood[11] or by removing one or both samples in different classes when they are nearest neighbors of each other[12].

ADASYN and SMOTE

Instead of just randomly oversampling there are also available approaches that generate new samples through the use of interpolation, such as SMOTE and ADASYN. However these methods can generate noisy samples so cleaning rule can be applied after oversampling[13].

Figure Extra: Other Sampling Approaches



Best Approach?

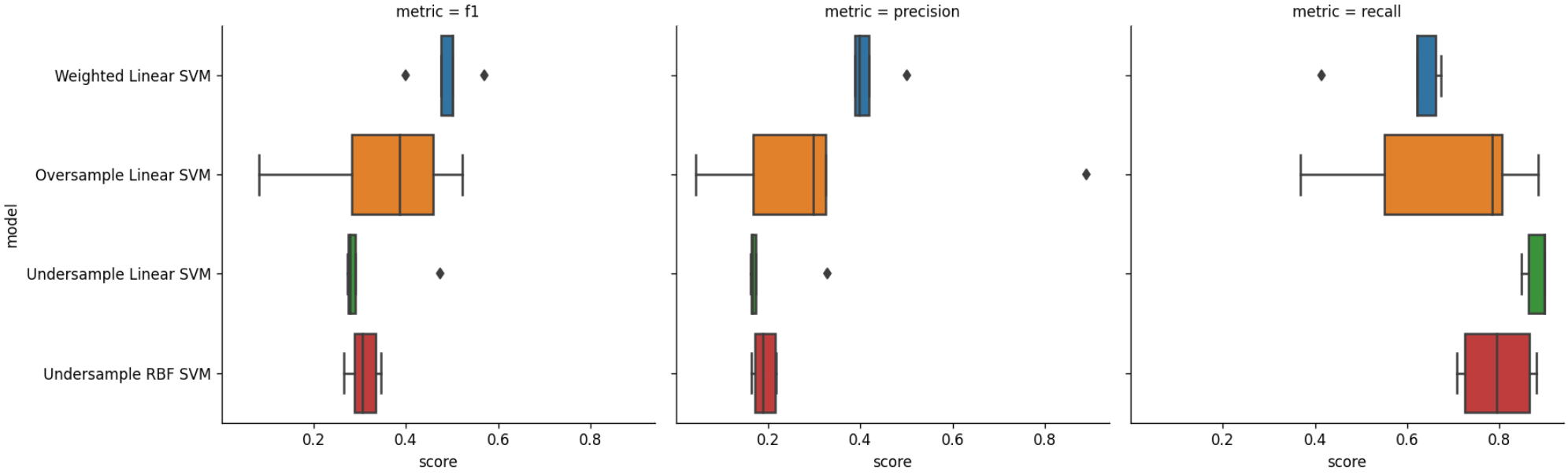
There is no one best approach, its typically dependent on the data and the aims for the model.

Below are examples of cross-validation scores for the best models (according to recall) for the different approaches.

Notes

- If over- and under-sampling are not much different (which in a lot of cases they won't be) you would typically favor undersampling for the savings you get in computational cost.

Figure 39: Best Recall Model Across Validation Folds



Using the figure above, for the client who wants the model to prioritise avoiding incorrectly classifying an individual who will default, we would probably choose the undersampled linear SVM.

As we can see on the test set, we get similar scores as we did on the validation.

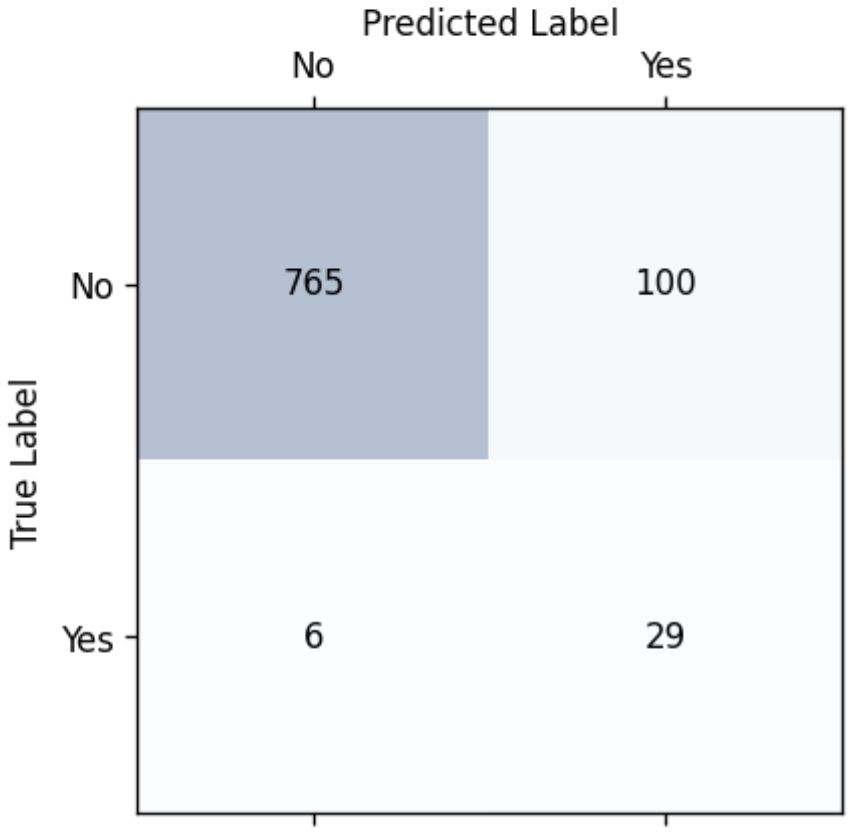
Notes

- *"The distinction between the training set, validation set, and test set is fundamentally important to applying machine learning methods in practice. Any choices made based on the test set accuracy "leak" information from the test set into the model. Therefore, it is important to*

keep a separate test set, which is only used for the final evaluation. It is good practice to do all exploratory analysis and model selection using the combination of a training and a validation set, and reserve the test set for a final evaluation—this is even true for exploratory visualization. Strictly speaking, evaluating more than one model on the test set and choosing the better of the two will result in an overly optimistic estimate of how accurate the model is."[16]

	No	Yes	avg / total
precision	0.99	0.25	0.96
recall	0.89	0.88	0.89
specificity	0.88	0.89	0.88
f1-score	0.94	0.39	0.91
geo	0.88	0.88	0.88
iba	0.78	0.78	0.78
support	958.00	42.00	1000.00

Figure 40: Test Set Performance



Extra: Improving the model using more features

Would adding in if someone is a student improve the model?

Note

- As noted above, you wouldnt normally do any more changes to your model after looking at the test set... but I did that for the lecture purposes!

default	student	balance	income

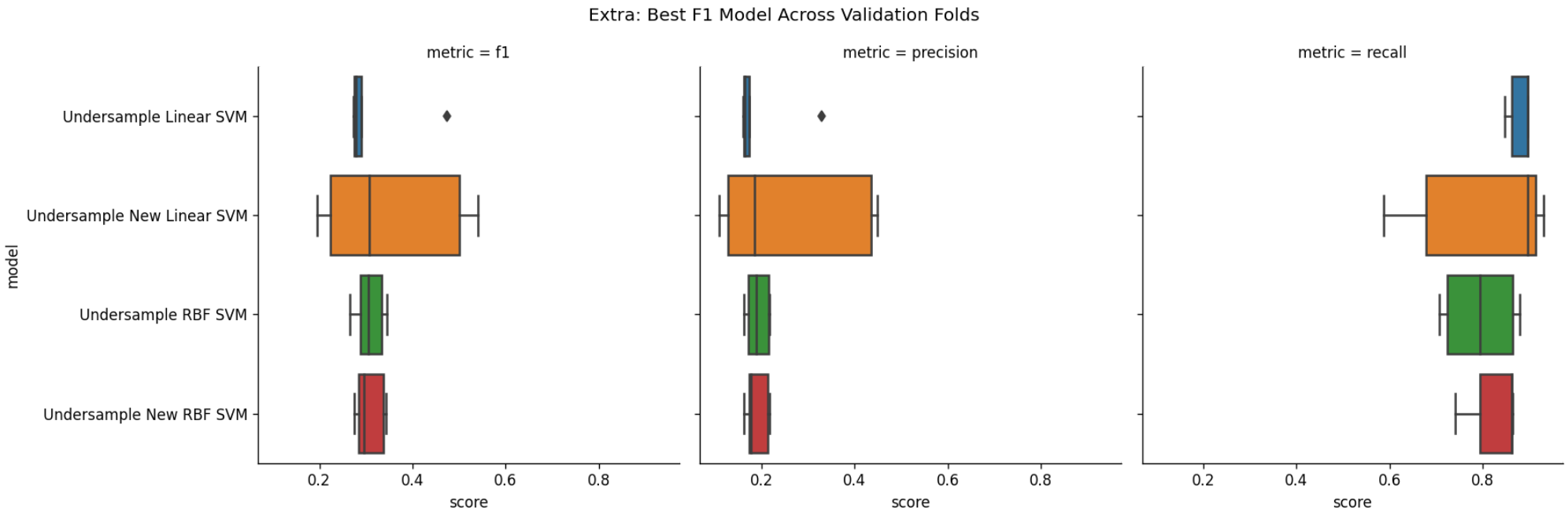
	default	student	balance	income
1	No	No	729.526495	44361.62507
2	No	Yes	817.180407	12106.13470
3	No	No	1073.549164	31767.13895
4	No	No	529.250605	35704.49394
5	No	No	785.655883	38463.49588

Note

- We want to put our `OneHotEncoder` into the pipeline but not do it for continous data

So in this case it does not seem to improve the metric of most interest (recall), although did improve precision at the expense of recall.

score			
metric	f1	precision	recall
model			
Undersample Linear SVM	0.318034	0.198239	0.879836
Undersample New Linear SVM	0.353487	0.261527	0.801110
Undersample New RBF SVM	0.306990	0.189444	0.824605
Undersample RBF SVM	0.307376	0.191444	0.793571



Fairness

Imagine we used the model in practice, and those deemed more likely to default were given more unfavourable terms due to them being seeming more risky according to the algorithm?

- Is it fair that ~1/8 people might be deemed risky, getting higher rates, and then not defaulting on the loan simply because of the algorithmic score given to them?
- Do we think income is a fair predictor for us to use, may it be related to gender? Are we now giving out less loans to women because they typically earn less?
- As this data comes from people we already gave loans to, do we know if the data is bias in any way (e.g. did we give more loans out to a particular group)?

Imagine adding their student status had improved our model. Would it be fair to judge their chances of defaulting based on their student status rather than their actual financial information?

- You could argue yes as students tend to have more "risky" financial behaviour and university is expensive.

- What about the responcible student who knows they can afford to pay it back?

Maybe we need to know more about the people applying for the loan, but what variables do we use?

- Postcode
  - Maybe you think people who live in wealthy areas are more likely to pay back the loan?
  - Doesn't this make it worse for people financially responcible people who live in poor areas?
  - Might this unfortunately be related to someones race/ethnicity?
- Criminal History
  - Maybe criminals are less likely to pay back loans?
  - Is our criminal system unbias?

This is a difficult but important aspect of ML and I leave it here to make you think about the possibilities. We'll be exploring this more in the last week.

Notes

- for more information on formal model comparisions see https://arxiv.org/pdf/1811.12808.pdf

7. Multi-Class

Some models (e.g. tree-based classifiers) are inherently multiclass, whereas other machine learning algorithms are able to be extended to multi-class classification using techniques such as the One-versus-Rest or One-versus-One methods[3].

One-verses-all (or One-vs-the-rest)

The *One-verses-all* approach is were you train a classifier for each class and select the class from the classifier that outputs the highest score[3].

In other terms, if we fit K SVMs, we assign a test observation, x^*, to the class for which $\beta_{0k} + \beta_{1k}x_1^*, \ldots, \beta_{pk}x_p^*$ is largest (the most confident)[5].

Advantage: As each class is fitted against all other classes for each classifier, it is relatively interpretable[14].

Disadvantages: Can result in ambiguious decision regions (e.g. could be class 1 or class 2), and classifiers could suffer from issues of class imballance[4].

Notes

- Alike to most plots in these lectures/notes, for ease of visualisation we'll only use two features.
- Although now inbuilt into `sklearn.svm.SVC`, you could also put the SVC inside `sklearn.multiclass.OneVsRestClassifier`

	param_svm_clf__class_weight	param_svm_clf__C	param_svm_clf__gamma	mean_test_score	std_test_score
23	balanced	42.715424	0.001768	0.966667	0.018856
50	balanced	11.649174	0.002483	0.966667	0.018856
25	None	106.523159	0.001016	0.966667	0.018856
51	balanced	2.137057	0.017867	0.966667	0.018856
6	balanced	0.397242	0.062924	0.966667	0.018856

OneVsOne

Another strategy is to use a *OneVsOne* approach.

This trains $N \times (N-1)/2$ classifiers by comparing each class against each other.

When a prediction is made, the class that is selected the most is chosen (*Majority Vote*)[3].

Advantage: It is useful where algorithms do not scale well with data size (such as SVM), because each training and prediction is only needed to be run on a small subset of the data for each classifer[3,14].

Disadvantages: Can still result in ambiguious decision regions and be computationally expensive[4].

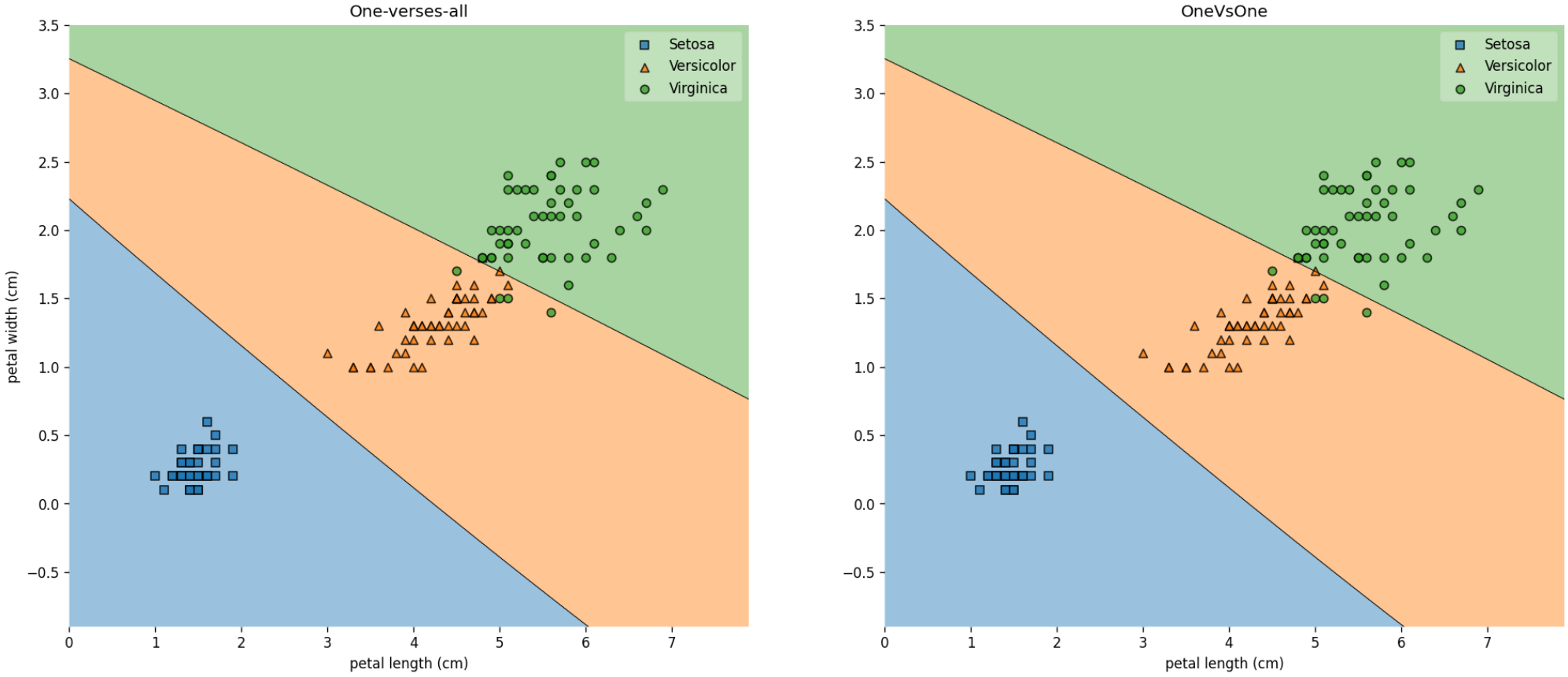
Notes

- Next week we'll talk more about Majority Voting ensembles for classification.

	param_svm_clf__class_weight	param_svm_clf__C	param_svm_clf__gamma	mean_test_score	std_test_score

	param_svm_clf__class_weight	param_svm_clf__C	param_svm_clf__gamma	mean_test_score	std_test_score
23	balanced	42.715424	0.001768	0.966667	0.018856
50	balanced	11.649174	0.002483	0.966667	0.018856
25	None	106.523159	0.001016	0.966667	0.018856
51	balanced	2.137057	0.017867	0.966667	0.018856
6	balanced	0.397242	0.062924	0.966667	0.018856

Figure 41: Multi-label classification on the Iris Dataset



Multiclass scoring metrics[9]

`scikit-learn` implements macro and micro averaging methods to extend scoring metrics to multiclass problems.

The *micro-average* is calculated from each TPs, TNs, FPs, and FNs of the system.

For example, the micro-average precision score for a k-class system is,

$$PRE_{micro} = \frac{TP_1 + \ldots + TP_K}{TP_1 + \ldots + TP_K + FP_1 + \ldots + FP_K}.$$

This is useful when we want to weight each instance or prediction equally.

The *macro-average* is the average scores of the different systems:

$$PRE_{macro} = \frac{PRE_1 + \ldots + PRE_K}{K}.$$

This is useful when we want to evaluate the overall performance of a classifier with regard to the most frequent class labels.

8. Strengths and Limitations

There are always advantages and disadvantages to using any model on a particular dataset.

Advantages

1. Kernels allow us to constuct hyperplanes in high dimensional spaces with tractable computation[6]
    - SVMs allow for complex decision boundaries on both low-dimensional and high-dimensional data (i.e., few and many features).

1. SVMs always converge on the same answer given identical data and hyper-parameters.

1. Kernelized support vector machines perform well on a variety of datasets[16].

1. By softening the margin using a budget (or cost) parameter (C), SVMs are relatively robust to outliers[17].

- Provided hyper-parameters are carefully selected, compared to some ML methods (e.g. Trees), SVM's are less likely to be effected by an over-representation of data in the training phase due to over-parameterization or over-fitting.

Disadvantages

1. Non-linear SVM's do not scale very well with the number of samples.
   - However, there are batch algorithms which do improve this[11].

1. SVMs require careful preprocessing of the data[16].
   - Tree-based models require little or no pre-processing.

1. SVM models are hard to inspect[16].
   - It can be difficult to understand why a particular prediction was made.
   - It might be tricky to explain the model to a nonexpert.

1. Discrete data typically presents a problem for SVMs.
   - However, there are alternative implimentations in the litriture to handle discrete data[10].

1. They are comparatively sensitive to hyperparamters, there can be quite a variation in score depending on setup.
   - The choice of kernel for example can change the results considerably[15].

1. SVMs only produce predicted class labels (and their *"confidence"*).
   - Obtaining predicted class probabilities requires additional adjustments and computations not covered here.

References

1. https://scikit-learn.org/stable/datasets/toy_dataset.html
2. Warwick J Nash, Tracy L Sellers, Simon R Talbot, Andrew J Cawthorn and Wes B Ford (1994) "The Population Biology of Abalone (*Haliotis* species) in Tasmania. I. Blacklip Abalone (*H. rubra*) from the North Coast and Islands of Bass Strait", Sea Fisheries Division, Technical Report No. 48 (ISSN 1034-3288)
3. Géron, A. (2017). Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.".

4. Murphy, K. P. (2012). Machine learning: a probabilistic perspective. MIT press.
5. James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. An introduction to statistical learning. Vol. 112. New York: springer, 2013.
6. http://contrib.scikit-learn.org/imbalanced-learn/stable/introduction.html
7. https://beckernick.github.io/oversampling-modeling/
8. Mani, I., & Zhang, I. (2003, August). kNN approach to unbalanced data distributions: a case study involving information extraction. In Proceedings of workshop on learning from imbalanced datasets (Vol. 126).
9. Raschka, Sebastian, and Vahid Mirjalili. Python Machine Learning, 2nd Ed. Packt Publishing, 2017.
10. Lemaître, G., Nogueira, F., & Aridas, C. K. (2017). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. The Journal of Machine Learning Research, 18(1), 559-563.
11. Wilson, D. L. (1972). Asymptotic properties of nearest neighbor rules using edited data. IEEE Transactions on Systems, Man, and Cybernetics, (3), 408-421.
12. Tomek, I. (1976). Two modifications of CNN. IEEE Trans. Systems, Man and Cybernetics, 6, 769-772.
13. Batista, G. E., Prati, R. C., & Monard, M. C. (2004). A study of the behavior of several methods for balancing machine learning training data. ACM SIGKDD explorations newsletter, 6(1), 20-29.
14. https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html
15. Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. Data mining and knowledge discovery, 2(2), 121-167.
16. Müller, A. C., & Guido, S. (2016). Introduction to machine learning with Python: a guide for data scientists. " O'Reilly Media, Inc.".
17. https://bradleyboehmke.github.io/HOML/svm.html

```
[NbConvertApp] Converting notebook 3_Applications.ipynb to html
[NbConvertApp] Writing 3972266 bytes to 3_Applications.html
[NbConvertApp] Converting notebook 3_Applications.ipynb to slides
[NbConvertApp] Writing 3380136 bytes to 3_Applications.slides.html
```