CLASSIFICATION METRICS

*Copyright Maven Analytics, LLC

CLASSIFICATION METRICS & EVALUATION

In this section we'll introduce **classification metrics for evaluating models**, discuss pros and cons, and demonstrate techniques for comparing model performance

TOPICS WE'LL COVER:

- Model Selection & Metrics
- F1 Score
- Soft Classification Thresholds
- Precision-Recall & F1 Curves
- ROC & AUC
- Multi-Class Metrics

GOALS FOR THIS SECTION:

- Calculate common metrics like accuracy, precision, recall and F1 using confusion matrices
- Learn why accuracy isn't always a useful metric for evaluating model performance
- Discuss when and why you might adjust the decision threshold for soft classification
- Visualize and interpret the ROC and Precision-Recall Curves to compare model performance

*Copyright Maven Analytics, LLC

MODEL SELECTION & METRICS

Model selection describes the process of training multiple models, comparing performance on Test data, and choosing the best option

- When comparing performance, it's important to prioritize the **most relevant metrics** based on the business context and the problem you are trying to solve

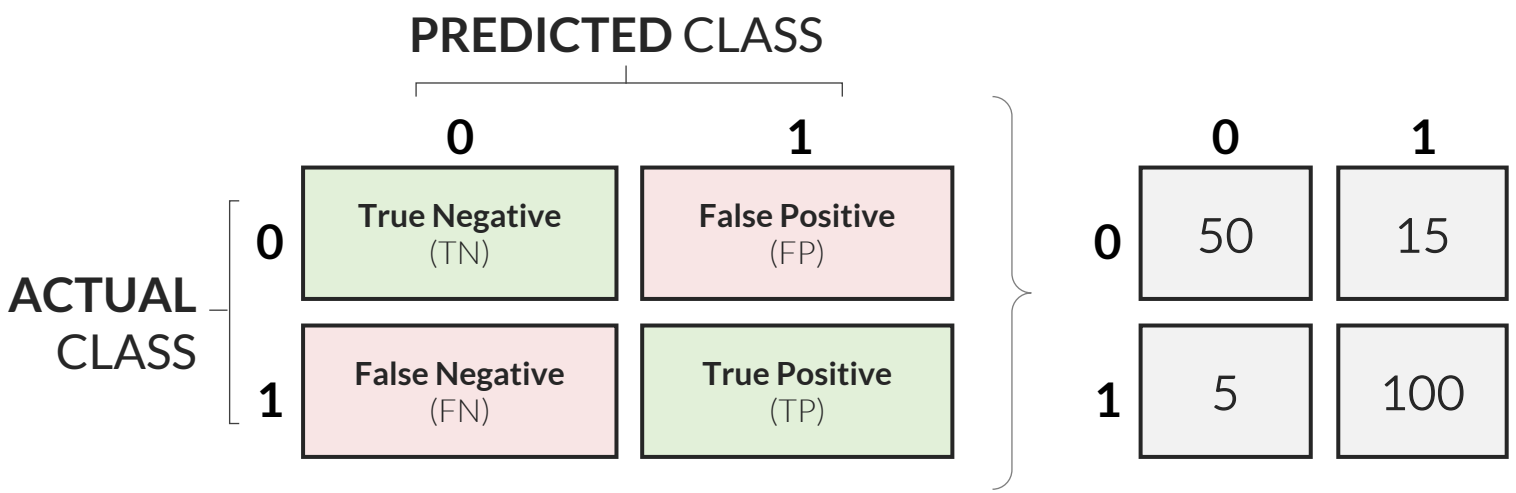
Metric	Definition
ACCURACY	The percentage of total observations classified correctly
PRECISION	The percentage of predicted positives classified correctly
RECALL	The percentage of actual positives that were correctly classified
F1	The harmonic mean of precision and recall
ROC AUC	Measures how well your model's predicted probabilities rank positive instances above negative instances for all decision thresholds

*Copyright Maven Analytics, LLC

RECAP: CONFUSION MATRIX

A **Confusion Matrix** is a table summarizing the frequency of **predicted** vs. **actual** classes for a given set of data

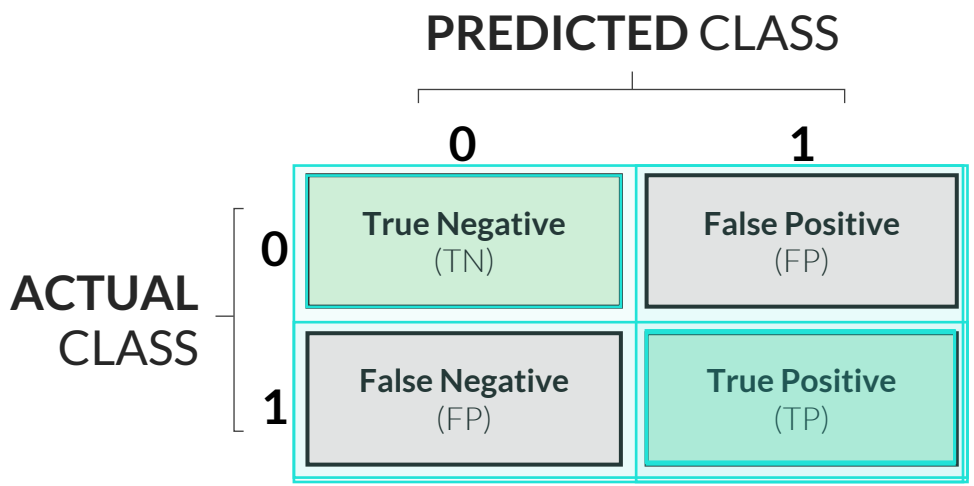
- This is the most common and concise way to evaluate performance and compare classification models against one another
- Confusion matrices can be used to derive several types of model performance metrics, including **accuracy**, **precision** and **recall**



*Copyright Maven Analytics, LLC

ACCURACY, PRECISION & RECALL

- Model Selection & Metrics
- F1 Score
- Soft Classification Thresholds
- Precision-Recall & F1 Curves
- ROC & AUC
- Multi-Class Metrics



Accuracy

(TP+TN) / (TP+TN+FP+FN)

*Of **all predictions**, what % were correct?*

Precision

TP / (TP+FP)

*Of all **predicted positives**, what % were correct?*

Recall

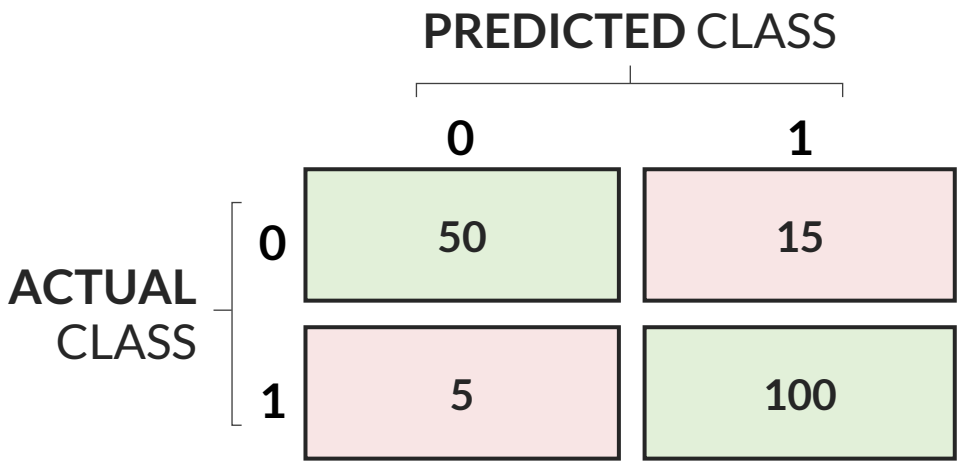
TP / (TP+FN)

*Of all **actual positives**, what % were predicted correctly?*

*Copyright Maven Analytics, LLC

ACCURACY, PRECISION & RECALL

- Model Selection & Metrics
- F1 Score
- Soft Classification Thresholds
- Precision-Recall & F1 Curves
- ROC & AUC
- Multi-Class Metrics



Accuracy

(TP+TN) / (TP+TN+FP+FN)

=

(100+50)/(100+50+5+15)

=

.88

Precision

TP / (TP+FP)

=

100/(100+15)

=

.87

Recall

TP / (TP+FN)

=

100/(100+5)

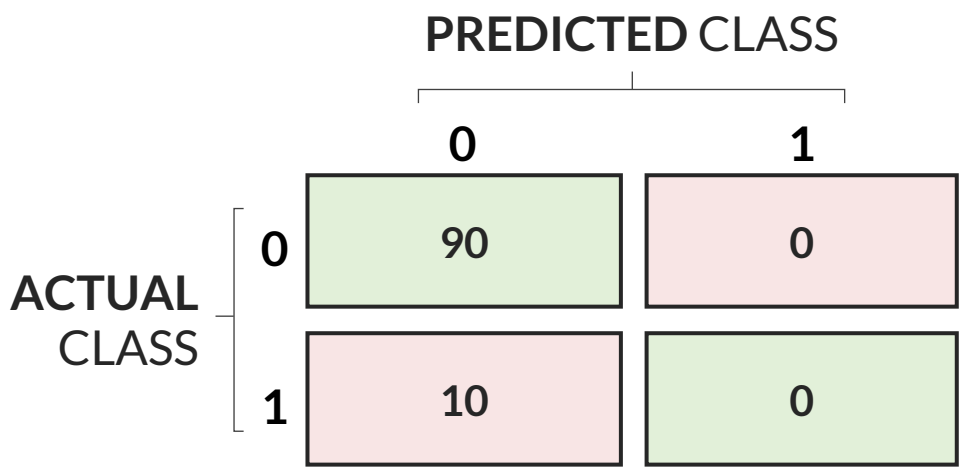
=

.95

*Copyright Maven Analytics, LLC

ACCURACY, PRECISION & RECALL

- Model Selection & Metrics
- F1 Score
- Soft Classification Thresholds
- Precision-Recall & F1 Curves
- ROC & AUC
- Multi-Class Metrics



Accuracy

(TP+TN) / (TP+TN+FP+FN)

=

(90+0)/(90+10+0+0)

=

.90

Precision

TP / (TP+FP)

=

0/(0+0)

=

No Precision

Recall

TP / (TP+FN)

=

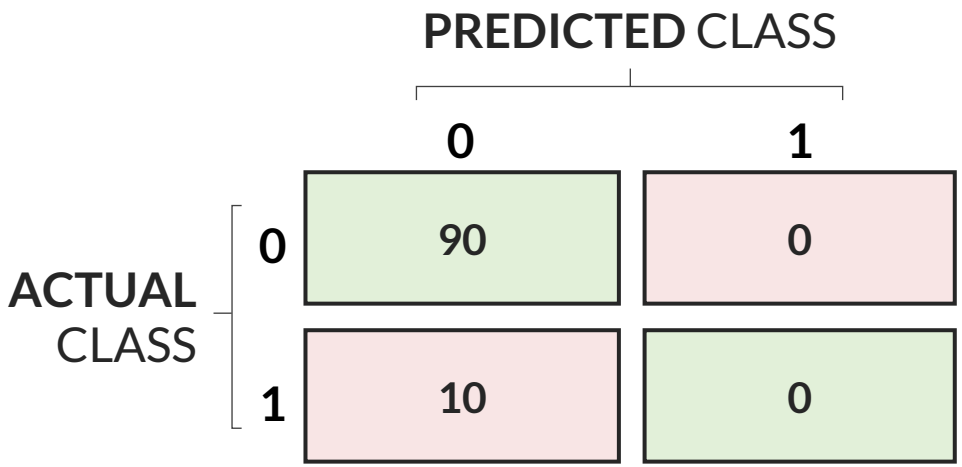
0/(0+10)

=

.00

*Copyright Maven Analytics, LLC

ACCURACY, PRECISION & RECALL



Accuracy alone **is not enough to properly evaluate a classification model**

- When **one class is rare**, we may have high accuracy but a useless model!
- **Precision** and **recall** add context, and may be superior to accuracy in some cases

*Copyright Maven Analytics, LLC

ACCURACY, PRECISION & RECALL IN PYTHON

Accuracy, Precision & Recall scores have sklearn functions associated with them

- They are available in the metrics module
- accuracy_score(actual_y, predicted_y),
- precision_score(actual_y, predicted_y)
- recall_score(actual_y, predicted_y)

```
from sklearn.metrics import accuracy_score, precision_score, recall_score

print(f"Test Accuracy: {accuracy_score(y_test, knn.predict(X_test_std))}")
print(f"Test Precision: {precision_score(y_test, knn.predict(X_test_std))}")
print(f"Test Recall: {recall_score(y_test, knn.predict(X_test_std))}")
```

```
Test Accuracy: 0.8875
Test Precision: 0.8108108108108109
Test Recall: 0.9375
```

*Copyright Maven Analytics, LLC

PRO TIP: F1 SCORE

F1 score measures the harmonic mean of precision and recall

- An F1 score of **1** indicates perfect precision & recall scores
- An F1 score near **0** indicates a significant imbalance between precision and recall
- sklearn.metrics.f1_score(y_actual, y_predicted)

$$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = 2\,\frac{precision * recall}{precision + recall}$$

*In mathematics this is known as a **harmonic mean,** which is favored for means of ratios as it ensures an equal weight on both ratios. It will thus be lower than the standard arithmetic mean.*

High F1

$$F_1 = 2\,\frac{.95 * .87}{.95 + .87} = \mathbf{.91}$$

$$F_1 = 2\,\frac{.5 * .9}{.5 + .9} = \mathbf{.64}$$

$$F_1 = 2\,\frac{0.1 * 1}{0.1 + 1} = \mathbf{.18}$$

Low F1

PRO TIP: F1 is a great metric for challenging modeling problems where a balance of precision and recall is critical

*Copyright Maven Analytics, LLC

ASSIGNMENT: MODEL METRICS

NEW MESSAGE

October 15, 2023

From: **Larry Logit** *(Sr. Statistician)*

Subject: **Model Metrics**

Hi there!

I need more information on your model's performance. Please calculate the accuracy, precision, recall, and F1 scores for your fitted model.

Thanks!

04_metrics_assignments.ipynb

Reply Forward

Key Objectives

1. Generate a confusion matrix and calculate F1, Precision, Recall & Accuracy on test data for a classification model

*Copyright Maven Analytics, LLC

SOFT CLASSIFICATION

By default, binary classification models will predict the positive class if the probability is >=**0.5** and the negative class if it's <**0.5**

- Tweaking the threshold can allow us to make more useful predictions based on the business context, without needing to refit or retrain the model

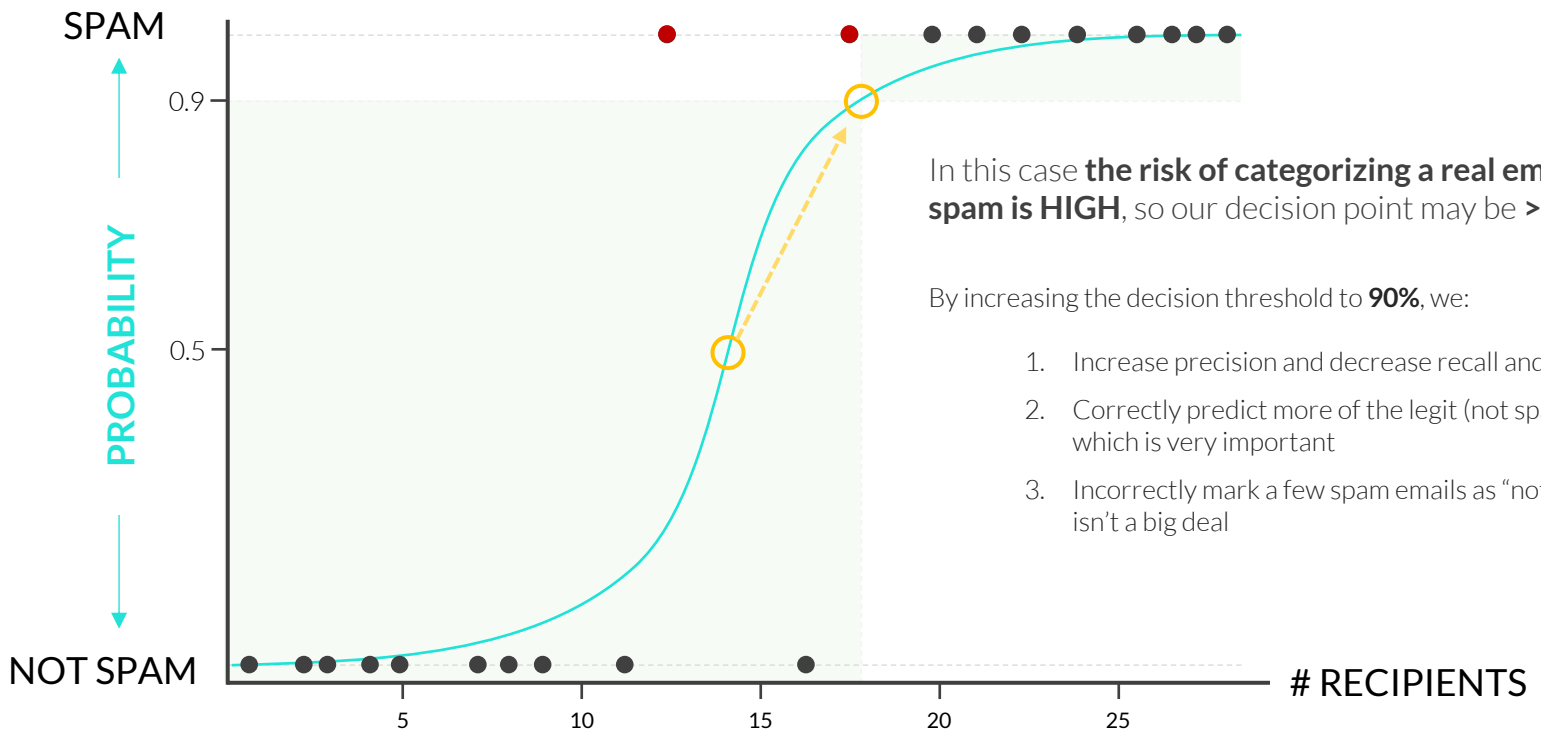
Choosing a threshold depends on the relative risk of a **false positive** compared to a **false negative**

- If the risk of a **false positive** is high, **increasing** the threshold may be appropriate
- If the risk of a **false negative** is high, **decreasing** the threshold may be appropriate

*Copyright Maven Analytics, LLC

SOFT CLASSIFICATION

EXAMPLE *Predicting if an email is spam based on the number of recipients*



In this case **the risk of categorizing a real email as spam is HIGH**, so our decision point may be **>50%**

By increasing the decision threshold to **90%**, we:

1. Increase precision and decrease recall and accuracy
2. Correctly predict more of the legit (not spam) emails, which is very important
3. Incorrectly mark a few spam emails as "not spam", which isn't a big deal

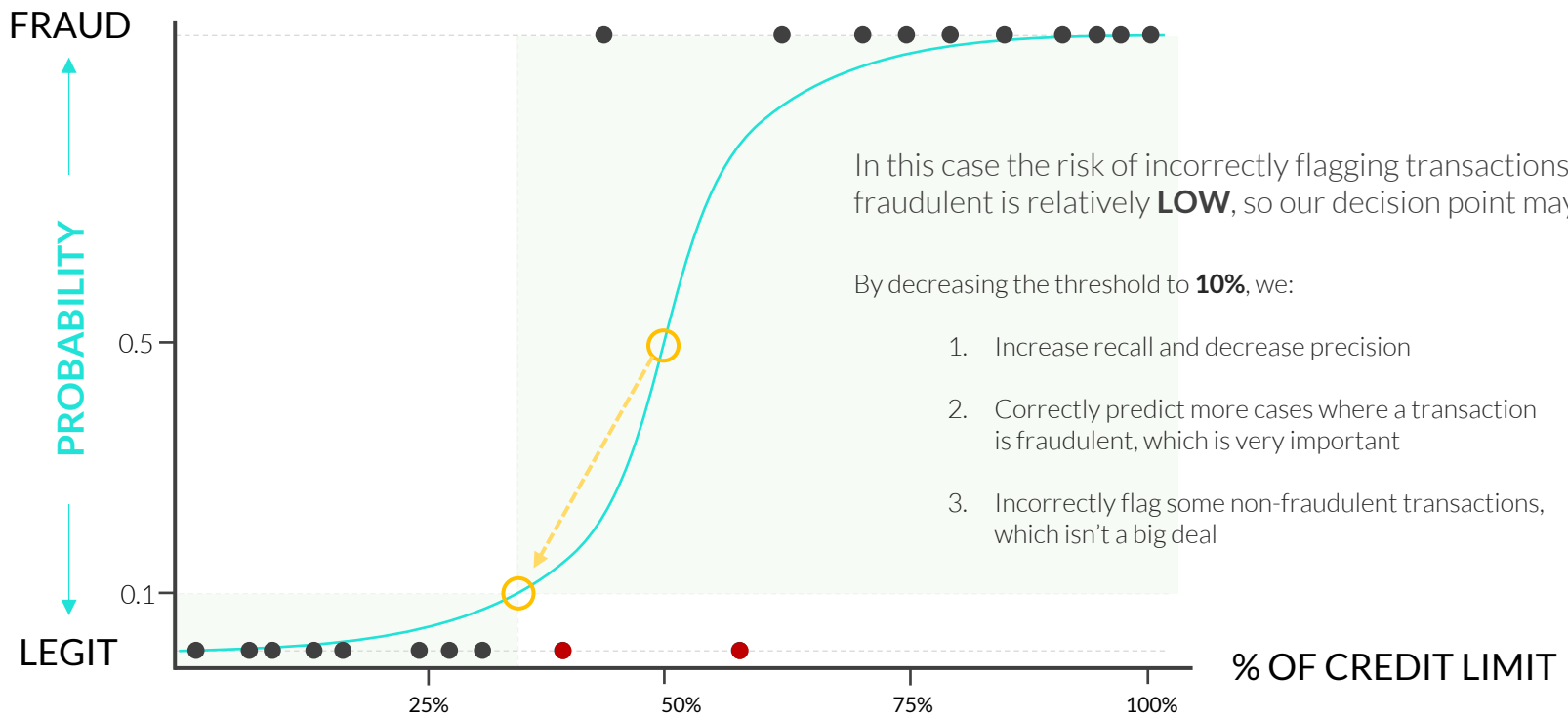
- In this case we're more worried about **false positives** (incorrectly categorizing real emails as spam), so a **higher** threshold helps us make those mistakes less often

*Copyright Maven Analytics, LLC

SOFT CLASSIFICATION

- Model Selection & Metrics
- F1 Score
- Soft Classification Thresholds
- Precision-Recall & F1 Curves
- ROC & AUC
- Multi-Class Metrics

EXAMPLE *Predicting if a credit card transaction is fraudulent*



- In this case we're more worried about **false negatives** (incorrectly predicting a transaction is legit), because it's less expensive to review potential fraud than it is to pay for a fraudulent transaction

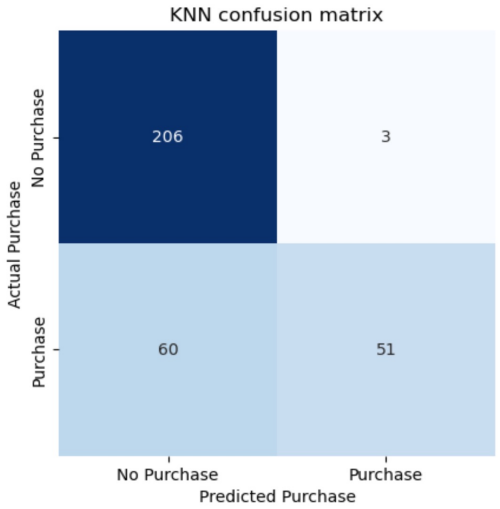
*Copyright Maven Analytics, LLC

SOFT CLASSIFICATION

- Model Selection & Metrics
- F1 Score
- Soft Classification Thresholds
- Precision-Recall & F1 Curves
- ROC & AUC
- Multi-Class Metrics

Threshold = **0.5**

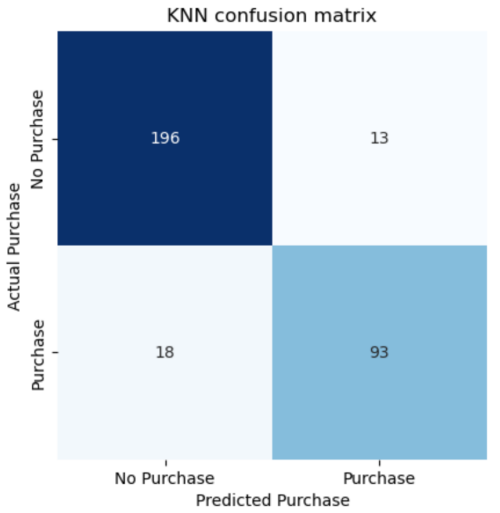
```
y_pred = knn.predict(X_train_std)
```



Accuracy: .803
Precision: .944
Recall: .449
F1: .618

Threshold = **0.32**

```
y_pred = knn.predict_proba(X_train_std)[:,1] > 0.32
```



Accuracy: .903 *(+0.1)*
Precision: .877 *(-0.07)*
Recall: .838 *(+0.39)*
F1: .857 *(+0.24)*

*By lowering our threshold, we were able to **correctly predict many more actual purchases**, increasing recall.*

This reduced precision slightly, but for problems where false negatives and false positives have an equal cost, this is a significant improvement!

*Copyright Maven Analytics, LLC

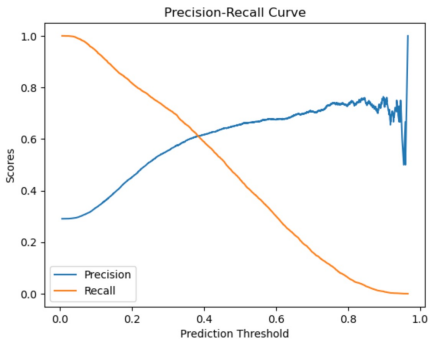
PRECISION-RECALL CURVE

The **precision-recall curve** helps optimize model thresholds by visualizing the trade-off between precision and recall for different threshold values

- In general, when we raise thresholds precision increases and recall decreases
- precision_recall_curve(y, lr.predict_proba(X)[:, 1])

```
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt

p_curve, r_curve, t_curve = precision_recall_curve(y_test, lr.predict_proba(X_test)[:,1])

plt.plot(t_curve, p_curve[:-1], label='Precision')
plt.plot(t_curve, r_curve[:-1], label='Recall')
plt.xlabel('Prediction Threshold')
plt.ylabel('Scores')
plt.legend()
plt.title('Precision-Recall Curve')
plt.show()
```

*The **precision_recall_curve** function outputs the precision curve, recall curve, and associated thresholds as NumPy arrays*



Our recall declines steadily as our threshold increases. This is perfect!

*Our precision generally increases as our threshold increases, but we see some declines starting at **0.8**, indicating our model probabilities are inaccurate in this range.*

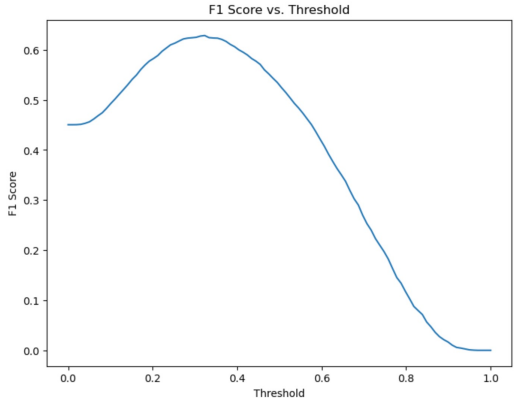
*Copyright Maven Analytics, LLC

F1 CURVE

The **F1 curve** helps optimize model thresholds by plotting the F1 score across different threshold values

- If we are tuning our model for F1, we want to pick the threshold associated with the peak

```
y_probs = lr.predict_proba(X_test)[:, 1]
thresholds = np.linspace(0, 1, 100)
f1_scores = [f1_score(y_test, (y_probs >= threshold)) for threshold in thresholds]

plt.figure(figsize=(8, 6))
plt.plot(thresholds, f1_scores)
plt.xlabel('Threshold')
plt.ylabel('F1 Score')
plt.title('F1 Score vs. Threshold')
plt.show()
```



*F1 is maximized at a threshold of **0.32***

This is close to, but not exactly where the precision and recall curves intersect.

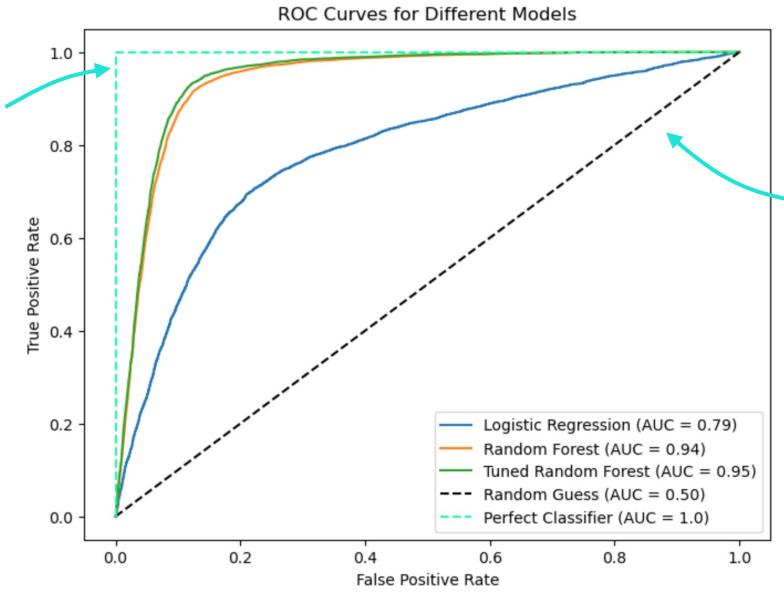
*Copyright Maven Analytics, LLC

ROC CURVE & AUC

The **ROC (Receiver Operating Characteristic) Curve** visualizes model performance for all thresholds, summarized by **AUC (Area Under the Curve)**

- AUC measures the probability that a randomly selected positive class has a higher predicted probability than a randomly selected negative class observation
- AUC falls between **0** and **1**, where **1** indicates a perfect classifier, **0** indicates a perfectly incorrect classifier, and **.5** indicates your model is no better than a random guess

A perfect model (AUC=1) captures 100% of positive class without a single false positive (the closer our model is to the corner, the better.



The diagonal line represents the performance of randomly assigning probabilities to positive and negative classes. For every true positive we predict correctly, we predict a false positive.

A fitted model cannot have a lower AUC than .5. Test AUC can fall below this, and if it does, it means your model isn't useful.

*Copyright Maven Analytics, LLC

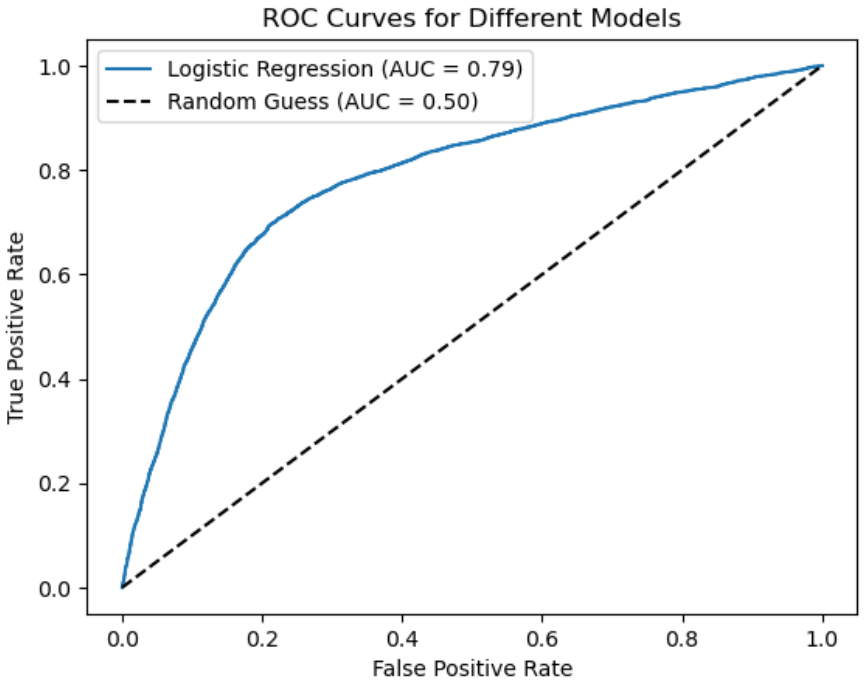
ROC CURVE & AUC

- Model Selection & Metrics
- F1 Score
- Soft Classification Thresholds
- Precision-Recall & F1 Curves
- ROC & AUC
- Multi-Class Metrics

Once you have a fitted model, you can plot the ROC curve and AUC metric based on your actual target values and predicted model probabilities

- fpr, tpr, thresholds = sklearn.metrics.roc_curve(actual_y, predicted_probabilities)
- auc_score = sklearn.metrics.auc(fpr, tpr)

```python
from sklearn.metrics import roc_curve, auc

logreg.fit(X_train, y_train)

# Predict probabilities for the positive class
y_probs = logreg.predict_proba(X_test)[:, 1]

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_probs)

# Calculate the AUC (Area Under the Curve)
auc_score = auc(fpr, tpr)

# Plot the ROC curve
plt.plot(fpr, tpr, label=f'Logistic Regression (AUC = {auc_score:.2f})')

# Draw Random Guess
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess (AUC = 0.50)')

# Modify Formatting
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves for Different Models')
plt.legend()
plt.show()
```



*Copyright Maven Analytics, LLC

CLASSIFICATION METRICS RECAP

Understanding **how** and **when** to use classification metrics is arguably as important as knowing the algorithms themselves

- If you use the wrong metric to guide the modeling process, **it doesn't matter which model you use**
- Aim to maximize 1-2 of these metrics, but they all add helpful context when assessing model performance

Metric	Definition	When to Use
ACCURACY	% of total observations classified correctly	Useful if your data is relatively balanced and you care about predicting positive and negative outcomes equally, or if the risk of each outcome is comparable
PRECISION	% of predicted positives classified correctly	Useful if false negatives aren't a big deal, but false positives are a major risk *(i.e. spam filter)*
RECALL	% of actual positives classified correctly	Useful if it's critical to predict ALL positive outcomes correctly, and false negatives are a major risk *(i.e. fraudulent transactions)*
F1	The harmonic mean of precision and recall	Useful if you have imbalanced data and need a balance between false positives and false negatives when predicting the positive class
ROC AUC	Measures how well your model's predicted probabilities rank positive instances above negative instances	Useful if you are most concerned with how well your probability scores are ranked (regardless of your decision threshold)

*Copyright Maven Analytics, LLC

ASSIGNMENT: THRESHOLD SHIFTING

NEW MESSAGE

October 16, 2023

From: **Larry Logit** *(Sr. Statistician)*

Subject: **Re: Model Metrics**

Hi there!

The model metrics look ok, but can you find the threshold that maximizes the F1 score? Please plot precision, recall and F1 vs the threshold and report the metrics as well as threshold value where F1 is maximized.

I'd also like to know what your model's AUC is and see the associated ROC curve.

Thanks!

04_metrics_assignments.ipynb

Reply Forward

Key Objectives

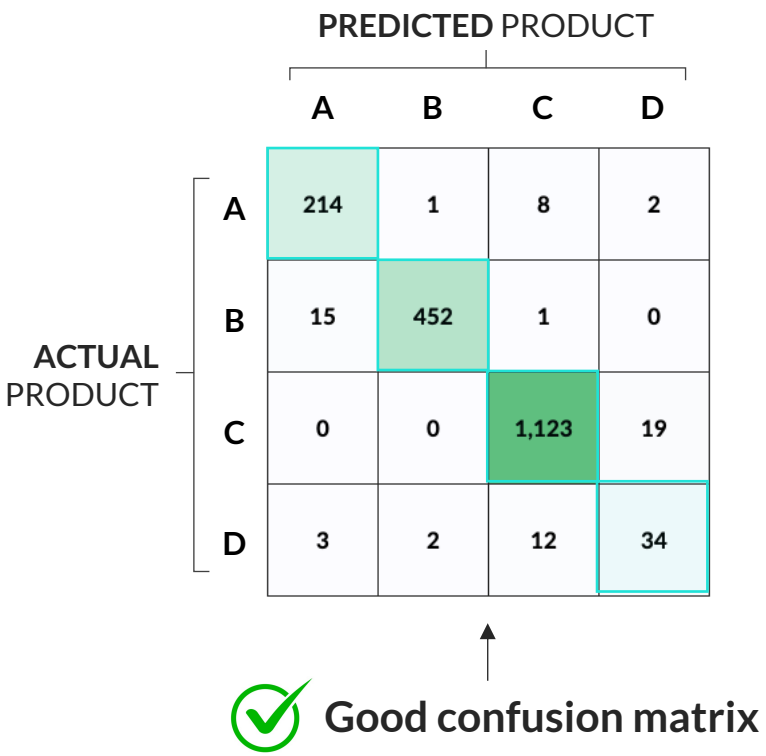
1. Plot Precision vs. Recall and F1 score based on threshold values
2. Change your model's threshold to maximize F1 Score and report final metrics
3. Plot an ROC curve and report AUC

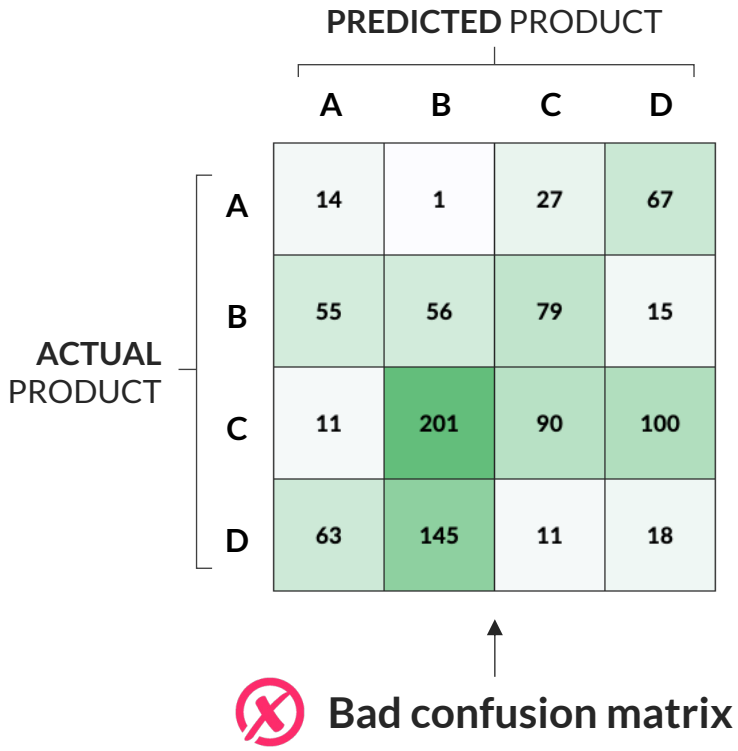
*Copyright Maven Analytics, LLC

MULTI-CLASS METRICS

You can use a Confusion Matrix to summarize **multi-class predictions** too

- A **multi-class confusion matrix** can also provide insight into relationships and interactions between classes, to help inform model improvements



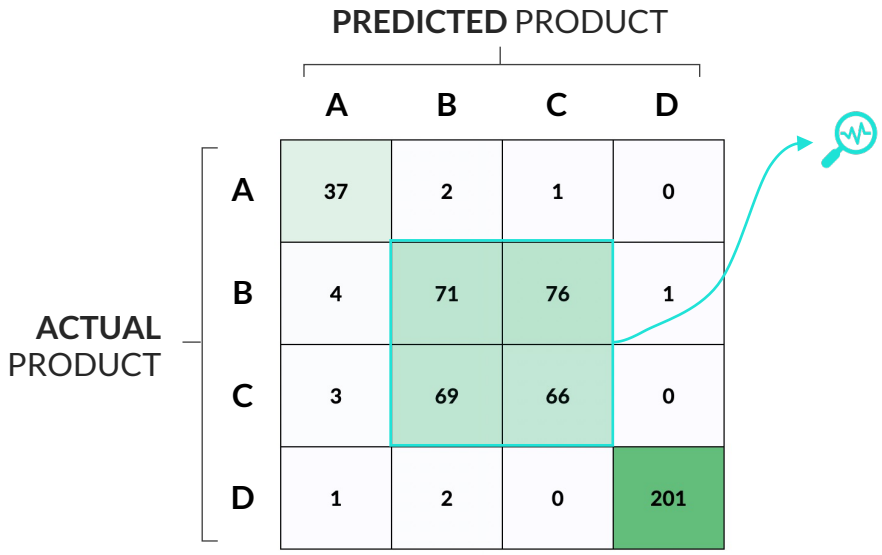


*Copyright Maven Analytics, LLC

MULTI-CLASS METRICS

You can use a Confusion Matrix to summarize **multi-class predictions** too

- A **multi-class confusion matrix** can also provide insight into relationships and interactions between classes, to help inform model improvements



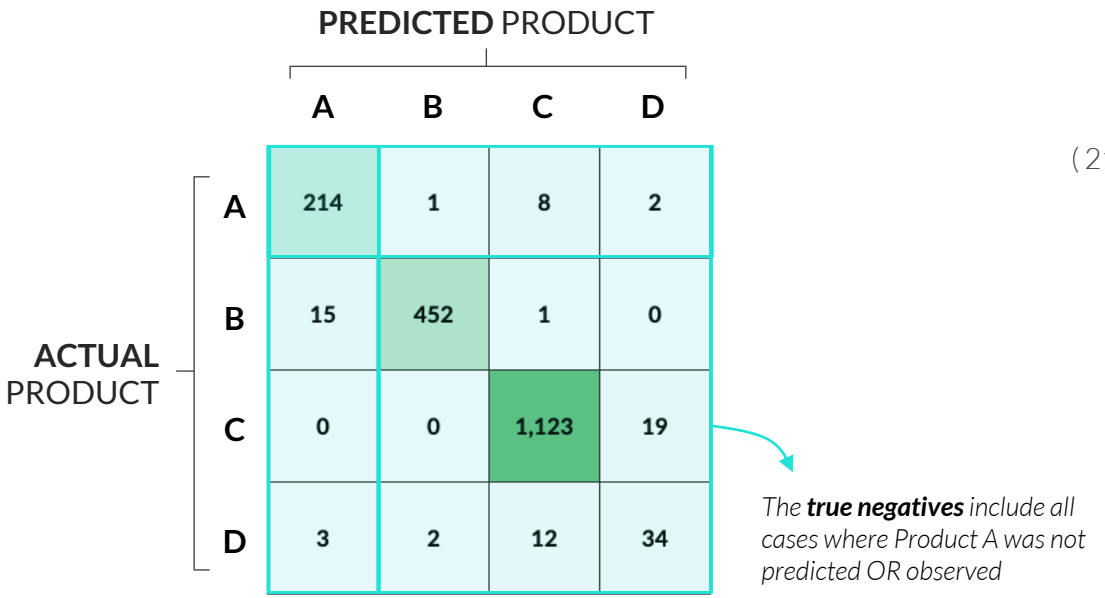
In this case our model has a hard time differentiating products **B** and **C**, often predicting the wrong one

- Are these products very similar? Do we need to group them or predict them separately?
- Can we engineer features to help distinguish B from C, and improve model accuracy?

*Copyright Maven Analytics, LLC

MULTI-CLASS METRICS

To score a **multi-class confusion matrix**, calculate metrics for each predicted class, then take a weighted average to evaluate the model as a whole



PRODUCT A:

Accuracy

(TP + TN) / (TP + TN + FP + FN)

(214 + 452 + 1 + 1123 + 19 + 2 + 12 + 34) / (1886)

=

.9846

Precision

TP / (TP + FP)

(214) / (214 + 15 + 3)

=

.9224

Recall

TP / (TP + FN)

(214) / (214 + 1 + 8 + 2)

=

.9511

*Copyright Maven Analytics, LLC

MULTI-CLASS METRICS

To score a **multi-class confusion matrix**, calculate metrics for each predicted class, then take a weighted average to evaluate the model as a whole

ACTUAL PRODUCT \ PREDICTED PRODUCT	A	B	C	D
A	214	1	8	2
B	15	452	1	0
C	0	0	1,123	19
D	3	2	12	34

PRODUCT B:

Accuracy

(TP + TN) / (TP + TN + FP + FN)

(452 + 214 + 8 + 2 + 1123 + 19 + 3 + 12 + 34) / (1886)

=

.9899

Precision

TP / (TP + FP)

(452) / (452 + 1 + 2)

=

.9934

Recall

TP / (TP + FN)

(452) / (452 + 15 + 1)

=

.9658

*Copyright Maven Analytics, LLC

MULTI-CLASS METRICS

To score a **multi-class confusion matrix**, calculate metrics for each predicted class, then take a weighted average to evaluate the model as a whole

ACTUAL PRODUCT \ PREDICTED PRODUCT	A	B	C	D
A	214	1	8	2
B	15	452	1	0
C	0	0	1,123	19
D	3	2	12	34

PRODUCT C:

Accuracy

(TP + TN) / (TP + TN + FP + FN)

(1123 + 214 + 1+ 2 + 15 + 452 + 3 + 2 + 34) / (1886)

=

.9788

Precision

TP / (TP + FP)

(1123) / (1123 + 8 + 1 + 12)

=

.9816

Recall

TP / (TP + FN)

(1123) / (1123 + 19)

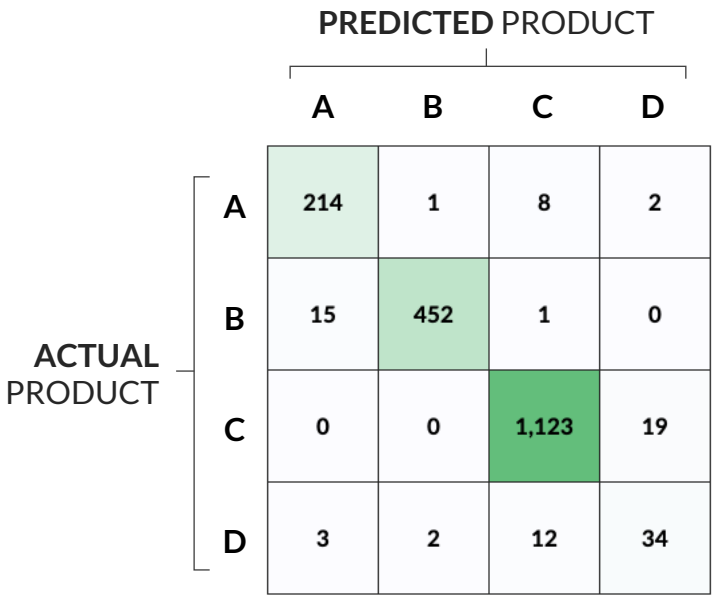
=

.9834

*Copyright Maven Analytics, LLC

MULTI-CLASS METRICS

To score a **multi-class confusion matrix**, calculate metrics for each predicted class, then take a weighted average to evaluate the model as a whole



PRODUCT D:

Accuracy

(TP + TN) / (TP + TN + FP + FN)

(34 + 214 + 1 + 8 + 15 + 452 + 1 + 1123) / (1886)

=

.9799

Precision

TP / (TP + FP)

(34) / (34 + 2 + 19)

=

.6182

Recall

TP / (TP + FN)

(34) / (34 + 3 + 2 + 12)

=

.6667

*Copyright Maven Analytics, LLC

MULTI-CLASS METRICS

To score a **multi-class confusion matrix**, calculate metrics for each predicted class, then take a weighted average to evaluate the model as a whole

214	1	8	2
15	452	1	0
0	0	1,123	19
3	2	12	34

	# Obs.	Accuracy	Precision	Recall
A	232	.9846	.9511	.9224
B	455	.9899	.9934	.9658
C	1,144	.9788	.9816	.9834
D	55	.9799	.6182	.6667
WEIGHTED AVG:		.9822	.9666	.9659

*Copyright Maven Analytics, LLC

MULTI-CLASS METRICS

- Model Selection & Metrics
- F1 Score
- Soft Classification Thresholds
- Precision-Recall & F1 Curves
- ROC & AUC
- Multi-Class Metrics

To score a **multi-class confusion matrix**, calculate metrics for each predicted class, then take a weighted average to evaluate the model as a whole

14	1	27	67
55	56	79	15
11	201	90	100
63	145	11	18

	# Obs.	Accuracy	Precision	Recall
A	143	.7500	.0979	.1284
B	403	.4795	.1390	.2732
C	207	.5498	.4348	.2239
D	200	.5792	.0900	.0759
WEIGHTED AVG:		.5563	.1868	.1994

*Copyright Maven Analytics, LLC

MULTI-CLASS METRICS IN PYTHON

- Model Selection & Metrics
- F1 Score
- Soft Classification Thresholds
- Precision-Recall & F1 Curves
- ROC & AUC
- Multi-Class Metrics

Sklearn metrics functions have built-in **multi-class functionality**

- Sklearn.metrics.precision_score(y_actual, y_pred, **average=[None, “macro”, “weighted”]**)

```
y_pred = lr.predict(X)

y_pred
```

```
array(['setosa', 'versicolor', 'setosa', 'virginica', 'setosa'],
      dtype=object)
```

average = None

Returns the metric of interest for each class

```
from sklearn.metrics import precision_score

precision_score(y, y_pred, average=None)
```

```
array([1.        , 0.97916667, 0.94230769])
```

average = “macro”

Returns a simple average of the metric

```
precision_score(y, y_pred, average="macro")
```

```
0.9738247863247862
```

average = “weighted”

Returns a weighted average for each class

```
precision_score(y, y_pred, average="weighted")
```

```
0.9738247863247864
```

In this example the weighted average is equivalent to the macro average because the classes were perfectly balanced, but “weighted” is ideal for imbalanced problems!

*Copyright Maven Analytics, LLC

ASSIGNMENT: MULTICLASS METRICS

NEW MESSAGE

October 18, 2023

From: **Larry Logit** *(Sr. Statistician)*

Subject: **Multi-Class Metrics**

Hi there!

Can you take another look at the multi-class credit model?

Please report the individual precision and recall scores for each class, as well as weighted averages for the model.

Thanks!

04_metrics_assignments.ipynb

Reply Forward

Key Objectives

1. Calculate metrics for a multi-class classification model, including individual class scores and weighted averages

*Copyright Maven Analytics, LLC

KEY TAKEAWAYS

Accuracy is the "default" performance metric, but it **isn't always useful**

- *Accuracy is suitable if you have balanced data and the cost of false positives & negatives is equal, but isn't ideal for imbalanced data or cases where false positives or negatives pose significant risk*

Precision, recall & F1 provide additional insight into performance

- *Recall should be prioritized if you want to avoid false negatives, precision should be prioritized if you want to avoid false positives, and F1 score is useful if you need a balance of both*

Adjusting decision thresholds may be necessary in some cases

- *If there is significant risk associated with incorrectly predicting a certain class (like misdiagnosing a serious medical condition), a higher or lower threshold may be appropriate (vs. the default 50%)*

ROC & AUC assess how well a model's probability scores are ranked

- *Because this metric is threshold agnostic, it can be very useful for comparing the overall performance of different models (a perfect classifier has an AUC of 1, and a random guess has an AUC of 0.5)*

*Copyright Maven Analytics, LLC

IMBALANCED DATA

*Copyright Maven Analytics, LLC

IMBALANCED DATA

In this section we'll cover techniques for **modeling imbalanced data**, which describes challenging problems where one class is much less common than the other(s)

TOPICS WE'LL COVER:

- Imbalanced Data
- Threshold Tuning
- Sampling Methods
- Oversampling
- SMOTE
- Undersampling
- Class Weights

GOALS FOR THIS SECTION:

- Introduce common types of imbalanced data and the challenges they pose for classification modeling
- Practice tuning decision thresholds to improve classification performance with imbalanced data
- Apply sampling methods for imbalanced data, including oversampling and undersampling
- Learn how to adjust class weights to force a model to prioritize accuracy predicting the minority class

*Copyright Maven Analytics, LLC

WHAT IS IMBALANCED DATA?

Imbalanced datasets describe classification data where **one class occurs less frequently than the other(s).**

- Usually, the minority class is much more valuable (or costly) than the majority class

Common Imbalanced Data Domains:

- E-Commerce Conversion
- Customer Churn
- Fraudulent Transactions
- Loan Defaults
- Disease Identification
- Product Defects

….and many more!

PRO TIP: When tackling an imbalanced classification problem on the job, speak with stakeholders about the costs/benefits of each quadrant of a confusion matrix to help guide your modeling approach

*Copyright Maven Analytics, LLC

WHAT IS IMBALANCED DATA?

Imbalanced datasets describe classification data where **one class occurs less frequently than the other(s).**

- These problems can pose significant challenges for modeling because the "best fit" for algorithms will often perform very poorly on rare classes

EXAMPLE

Predicting Fraudulent Credit Card Transactions

V25	V26	V27	V28	Amount	Fraud_Flag
0.128539	-0.189115	0.133558	-0.021053	149.62	0
0.167170	0.125895	-0.008983	0.014724	2.69	0
-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
0.647376	-0.221929	0.062723	0.061458	123.50	0
-0.206010	0.502292	0.219422	0.215153	69.99	0

```
cc_df["Fraud_Flag"].value_counts()
```

```
0    284315
1       492
Name: Fraud_Flag, dtype: int64
```

```
cc_df["Fraud_Flag"].value_counts(normalize=True)
```

```
0    0.998273
1    0.001727
Name: Fraud_Flag, dtype: float64
```

*Less than **2 in 1000** transactions were fraudulent, so our model could achieve 99.8% accuracy without predicting a single instance of fraud!*

There's no universal definition on exactly how rare a class must be to be "imbalanced ", but when working with binary classification **you should be prepared to leverage imbalanced techniques if one class is <30% of your data**

*Copyright Maven Analytics, LLC

MANAGING IMBALANCED DATA

- Imbalanced Data
- Threshold Tuning
- Sampling Methods
- Oversampling
- SMOTE
- Undersampling
- Class Weights

Several strategies can be used to help **manage imbalanced datasets**

- Selecting the correct metric *(precision, recall, etc.)* is especially important for these problems

1 Do Nothing

Sometimes your model will perform well without special treatment – try fitting a baseline model first!

2 Tune the decision threshold

Shifting the threshold can often improve model performance without the need for more advanced methods

3 Apply sampling methods

Create a more balanced training dataset by removing negative class observations, or duplicating/synthesizing positive class obvservations

4 Adjust class weights in the model cost function

Instruct the model to penalize mistakes on the rare class more heavily, improving metrics like precision & recall

*Copyright Maven Analytics, LLC

MANAGING IMBALANCED DATA

- Imbalanced Data
- Threshold Tuning
- Sampling Methods
- Oversampling
- SMOTE
- Undersampling
- Class Weights

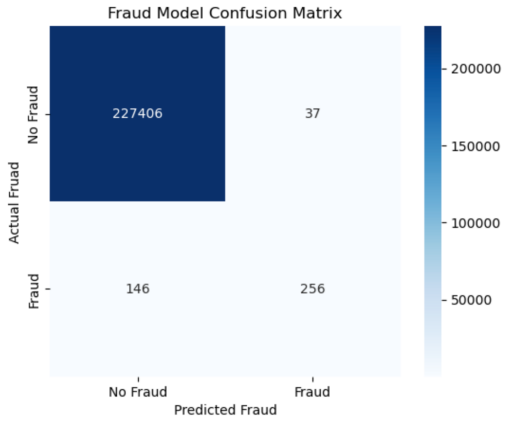
Models on imbalanced data often have **high accuracy but low precision & recall**

- Sometimes your model will perform well without the use of imbalanced data techniques, so don't jump straight to advanced techniques without getting a baseline first!

```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression(max_iter=1000)

lr = logreg.fit(X_train, y_train)

print(f"Train Accuracy: {lr.score(X_train, y_train)}")
print(f"Test Accuracy: {lr.score(X_test, y_test)}")
```

```
Train Accuracy: 0.9991968224011938
Test Accuracy: 0.9991397773954567
```

*Accuracy is extremely high, but this doesn't tell us much given **99.8%** of our data is not fraudulent*




```
print(f"Training Precision: {round(precision_score(y_train, y_pred), 2)}")
print(f"Training Recall: {round(recall_score(y_train, y_pred), 2)}")
```

```
Training Precision: 0.87
Training Recall: 0.64
```

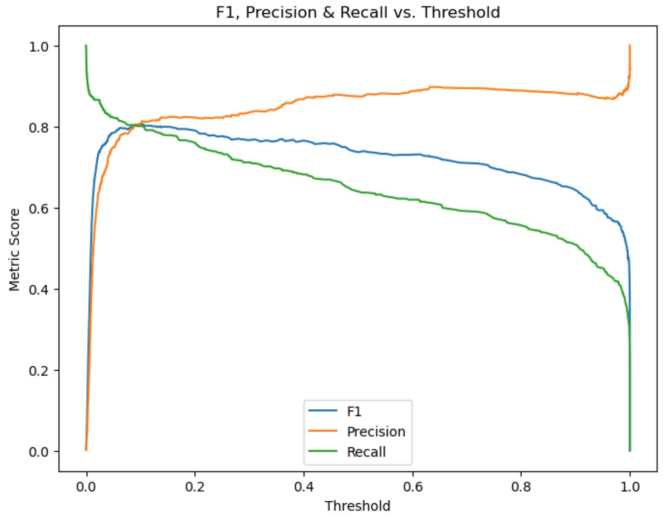
Our precision & recall aren't too bad given the extreme imbalance! But we can likely improve recall without sacrificing too much precision if we shift our threshold…

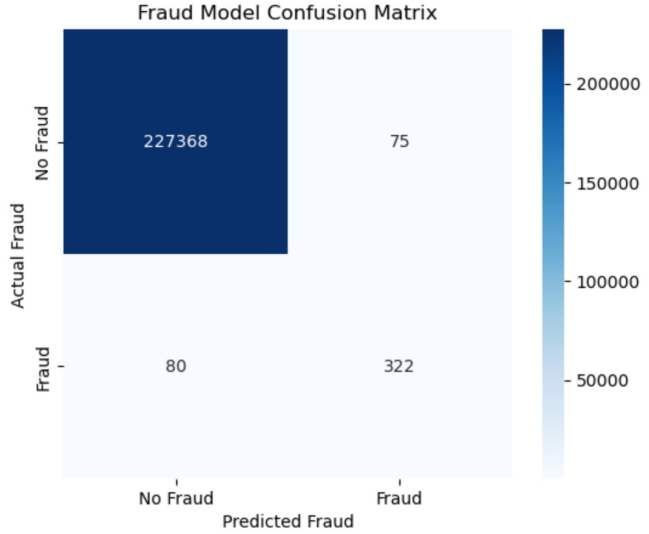
*Copyright Maven Analytics, LLC

THRESHOLD TUNING

- Imbalanced Data
- Threshold Tuning
- Sampling Methods
- Oversampling
- SMOTE
- Undersampling
- Class Weights

Tuning the model's decision threshold can improve your metric of interest





*We can increase recall from **.64** to **.80** by shifting our threshold close to .1! Our precision drops from .87 to .80, but this is likely worth it*

Suppose it costs **$5** for a human to review a transaction flagged by our model, and a fraudulent transaction costs the business **$122**

- This means a true positive is worth **$117** and a false positive costs **$5**
- Our old matrix yields a net benefit of ($117*256) – ($5*37) = **$29,767**
- Our new matrix yields a net benefit of ($117*322) – ($5*75) = **$37,299**

*Copyright Maven Analytics, LLC

SAMPLING METHODS

Sampling methods change the class distribution in the training data, which can lead to a model that better captures the positive class

- No method is perfect and each has pros & cons based on the data you're working with, but we can always leverage cross-validation to determine the optimal approach

Oversampling

Randomly duplicate some or all minority class observations to improve training data class balance

SMOTE (Synthetic Minority Oversampling Technique)

Advanced technique that generates synthetic datapoints using a distance-based approach

Undersampling

Randomly remove majority class observations to improve training data class balance

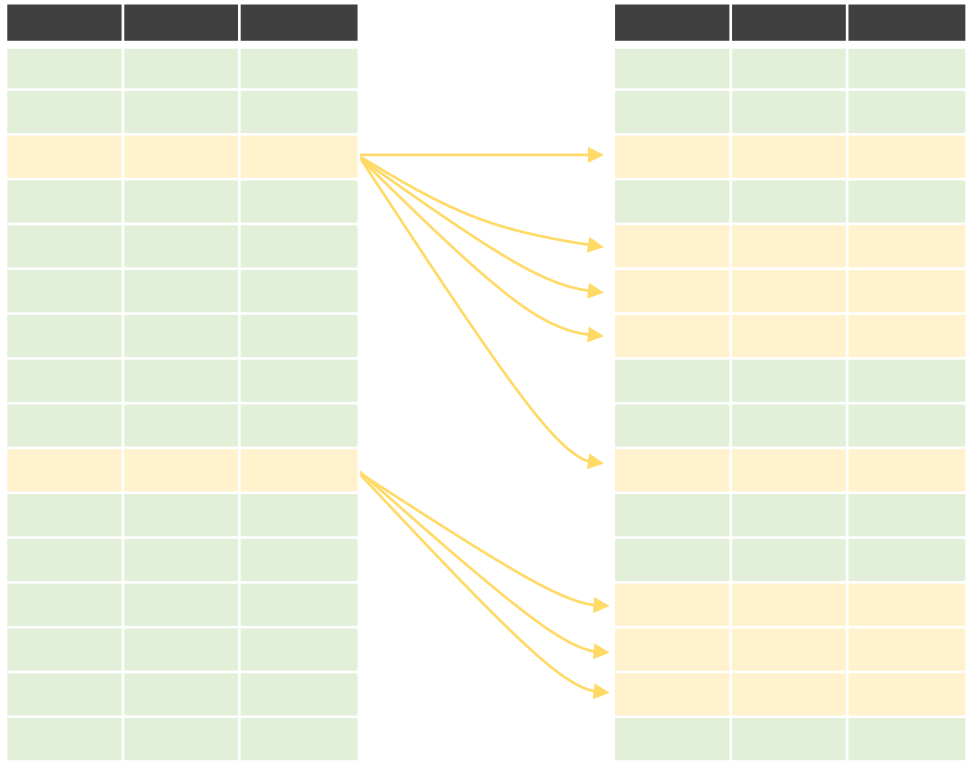
When using these sampling methods, **DO NOT apply them to your validation and test data**; out-of-sample data should reflect real world observations to ensure an accurate assessment of performance

*Copyright Maven Analytics, LLC

OVERSAMPLING

- Imbalanced Data
- Threshold Tuning
- Sampling Methods
- Oversampling
- SMOTE
- Undersampling
- Class Weights

Oversampling (or "upsampling") increases the number of minority class observations in the training dataset by duplicating actual observations or creating synthetic samples



PROS:

- ✓ Gives the model more opportunity to learn patterns associated with the minority class
- ✓ Retains all observations from original data

CONS:

- Duplicating observations means your model may overfit to specific minority class cases
- Increases the size of your data, which can tax resources if your data is already large

*Copyright Maven Analytics, LLC

OVERSAMPLING IN PYTHON

We can use the **imblearn library** for oversampling in Python

- imblearn.over_sampling.RandomOverSampler(sampling_strategy, random_state)

```
import imblearn.over_sampling as OS

n_pos = np.sum(y_train == 1)
n_neg = np.sum(y_train == 0)

# create 4x as many positive samples
ratio = {1 : n_pos * 4, 0 : n_neg}

# randomly oversample positives
ROS = OS.RandomOverSampler(
    sampling_strategy = ratio,
    random_state=2023
)

X_train_rs, y_train_rs = ROS.fit_resample(X_train, y_train)

print(f"Original Positive Class Count: {np.sum(y_train)}")
print(f"Oversample Positive Class Count: {np.sum(y_train_rs)}"
```

```
Original Positive Class Count: 402
Oversample Positive Class Count: 1608
```

*Here we're specifying we want to increase the minority class size by **4 times**, while keeping the majority class the same*

Data	Test F1
Original Data	.671
Oversampled Data	**.753**

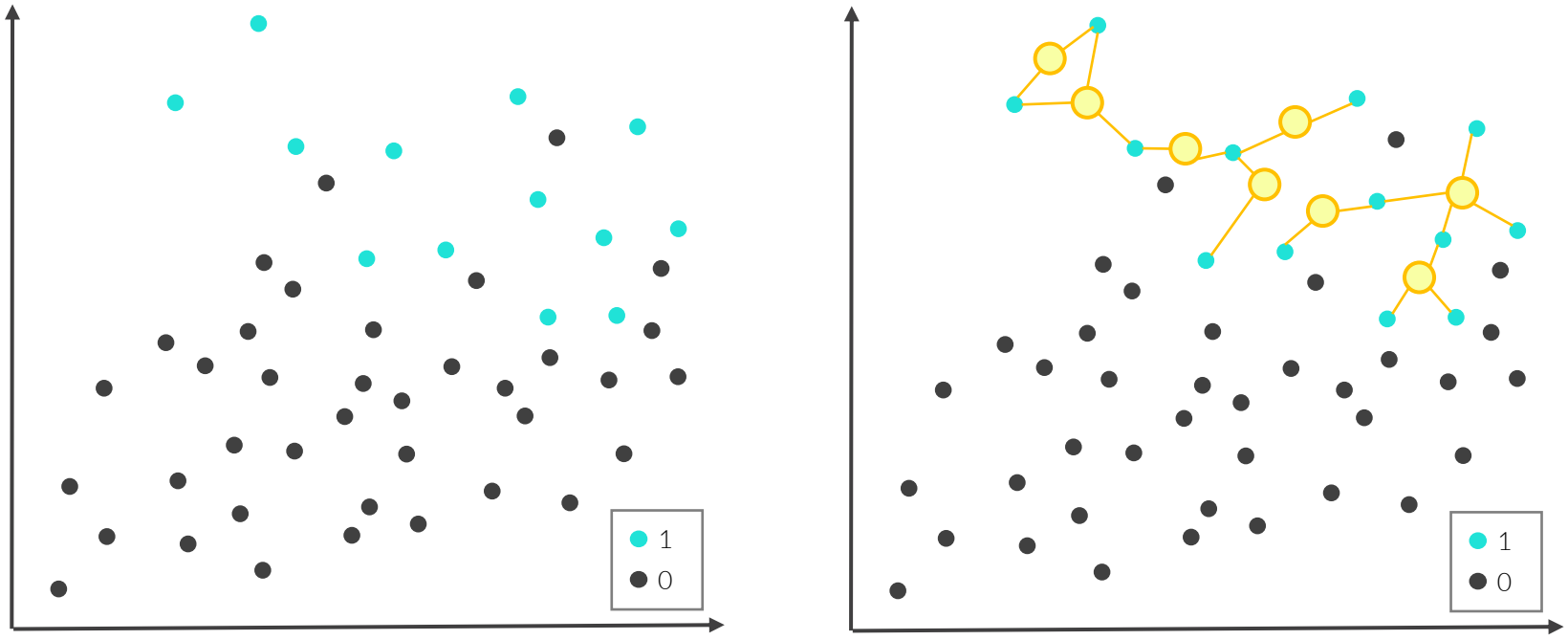
*Logistic Regression Test F1 increased from **.671** to **.753** when fit on oversampled data! We could further tune this by trying different oversampling ratios.*

*Copyright Maven Analytics, LLC

SMOTE

- Imbalanced Data
- Threshold Tuning
- Sampling Methods
- Oversampling
- SMOTE
- Undersampling
- Class Weights

SMOTE (Synthetic Minority Oversampling Technique) is a distance-based algorithm that creates synthetic minority class observations near existing ones



Because new rows are created rather than duplicated, **SMOTE is less prone to overfitting.** The downside is that it's more computationally expensive and relies on KNN to create neighboring samples, which may not be accurate.

*Copyright Maven Analytics, LLC

SMOTE IN PYTHON

We can use the **imblearn library** for SMOTE in Python

- imblearn.over_sampling.SMOTE(sampling_strategy, random_state)

```
import imblearn.over_sampling

n_pos = np.sum(y_train == 1)
n_neg = np.sum(y_train == 0)

# create 4x as many positive samples
ratio = {1 : n_pos * 4, 0 : n_neg}

# SMOTE oversampling for positives
smt = imblearn.over_sampling.SMOTE(
    sampling_strategy = ratio,
    random_state=2023
)

X_train_rs, y_train_rs = smt.fit_resample(X_train, y_train)
```

Data	Test F1
Original Data	.671
Oversampled Data	.753
SMOTE Data	**.760**

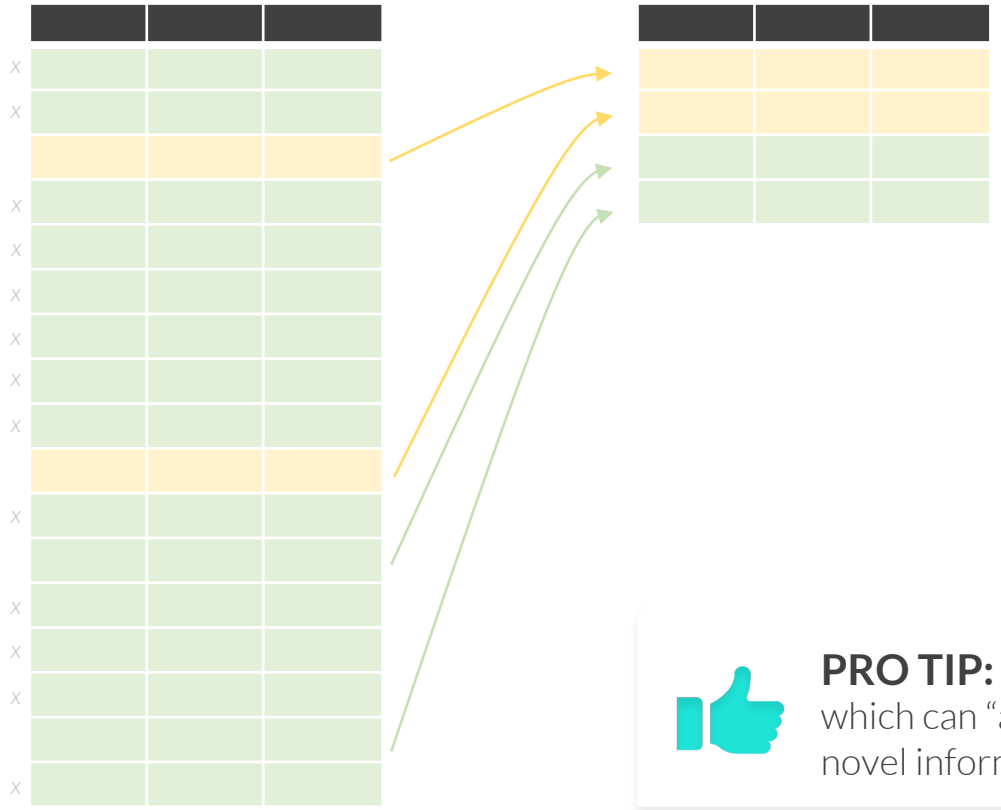
The synthetic samples performed even better than duplicated values!

*Copyright Maven Analytics, LLC

UNDERSAMPLING

- Imbalanced Data
- Threshold Tuning
- Sampling Methods
- Oversampling
- SMOTE
- Undersampling
- Class Weights

Undersampling, (or "downsampling") decreases the number of majority class observations in the training data by randomly removing majority class rows



PROS:

- ✓ Can improve model fit by making data more balanced
- ✓ Reduces data volume and computational costs

CONS:

- Throws out information that may have helped improve model fit

PRO TIP: Undersampling is best for larger datasets, which can "afford" to lose rows without throwing out novel information

UNDERSAMPLING IN PYTHON

We can use the **imblearn library** for undersampling in Python

- imblearn.under_sampling.RandomUnderSampler(sampling_strategy, random_state)

```
import imblearn.under_sampling as US

# Define Minority Class %
minority_pct = .30

# randomly undersample negative samples:
RUS = US.RandomUnderSampler(
                    sampling_strategy = (minority_pct)/(1 - minority_pct),
                    random_state=2023
)

X_train_rs, y_train_rs = RUS.fit_resample(X_train, y_train)

print(f"Original Negative Class Count: {np.mean(y_train)}")
print(f"Undersample Negative Class Count: {np.mean(y_train_rs)}")
```

```
Original Negative Class Count: 0.0017643573481972393
Undersample Negative Class Count: 0.30022404779686335
```

Here we're specifying we want to randomly discard majority rows to create a final training set with a ***30/70 minority/majority split***

Data	Test F1
Original Data	.671
Oversampled Data	.753
SMOTE Data	.760
Undersampled Data	**.149**

F1 with undersampled data performed much worse – we can try tuning the ratio further!

ASSIGNMENT: SAMPLING STRATEGIES

NEW MESSAGE

October 20, 2023

From: **Larry Logit** *(Sr. Statistician))*

Subject: **Imbalanced Techniques**

Hi there!

Let's try applying some imbalanced sampling methods to our income data.

It wasn't too imbalanced, but we may be able to get a bit better performance here.

Thanks!

05_imbalanced_data_assignments.ipynb

Reply Forward

Key Objectives

1. Use oversampling, undersampling, and SMOTE on your data to determine if model performance is improved
2. Tune your model threshold on your best imbalanced data model to optimize F1

*Copyright Maven Analytics, LLC

CHANGING CLASS WEIGHTS

- Imbalanced Data
- Threshold Tuning
- Sampling Methods
- Oversampling
- SMOTE
- Undersampling
- Class Weights

By default, **classes are weighted equally** when fitting a model to your data

- You can **tune the weighting hyperparameter** to increase the cost for missing on the minority class, which can improve accuracy on the positive class
- Class weighting can be combined with other methods like threshold tuning or sampling

```
# default equal weight model
lr = LogisticRegression()

# rebalance the weighting so the 'total' weight of each class is equal
lr_balanced = LogisticRegression(class_weight='balanced')

# intermediate weight (minority class has 4x weight of majority)
lr_4x = LogisticRegression(class_weight={1: 4, 0: 1})
```

```
# equally weighted model (default)
lr.fit(X_train, y_train)

# multiplies minority by common:rare ratio
lr_balanced.fit(X_train, y_train)

# intermediate weight (4x minority)
lr_4x.fit(X_train, y_train)
```

Weighting	Test F1	Test AUC
Default	.671	.974
Balanced	.117	**.982**
4x	**.757**	.979

Our balanced model had the best AUC, but F1 got worse. Changing the decision threshold could yield a better F1, but given how close our 4x model's AUC was, I would choose the 4x model here.

*Copyright Maven Analytics, LLC

ASSIGNMENT: CLASS WEIGHTS

NEW MESSAGE

October 20, 2023

From: **Larry Logit** *(Sr. Statistician))*

Subject: **Re: Imbalanced Techniques**

Hi there!

The sampling work you did was very helpful, thanks!

I forgot to mention I also want to try changing the class weights of the logistic model to see if that helps as well.

Let me know how it goes.

Thanks!

Larry

05_imbalanced_data_assignments.ipynb

Reply Forward

Key Objectives

1. Change the class weights of your model to maximize AUC
2. Tune the model threshold to maximize the F1 Score

*Copyright Maven Analytics, LLC

IMBALANCED DATA RECAP

1. **Do Nothing**

   Establish a baseline level of performance first – your model may perform well without specialized techniques!

2. **Tune the decision threshold**

   Threshold tuning is worth trying in tandem with all techniques listed. The default threshold of .5 is rarely the best, especially for imbalanced data

3. **Apply sampling methods**

   Sampling methods can help improve performance if your baseline model performance is poor. Oversampling and SMOTE should be used for small-medium data, while undersampling is a viable option for larger datasets.

4. **Adjust class weights in the model cost function**

   Changing the class weights is relatively easy to implement and doesn't change the underlying distribution of data. In the most challenging cases, this is often combined with sampling strategies.

*Copyright Maven Analytics, LLC

KEY TAKEAWAYS

Imbalanced data poses unique challenges to classification models

- *More often than not, the rare class is the one you need to predict accurately!*

Threshold shifting can improve precision and recall for the minority class

- *This technique should be used regardless of which other methods you use*

Sampling methods change the class distribution in your training data

- *Oversampling and SMOTE add duplicate or synthetic positive class observations to your data, undersampling removes negative class observations*

Adjust class weights to prioritize accuracy on the minority class

- *This is a model hyperparameter that you can tune via cross validation*

*Copyright Maven Analytics, LLC