Exercise: Imbalanced data model bias

In this exercise, we'll take a closer look at *imbalanced datasets*, what effects they have on predictions, and how they can be addressed.

We'll also employ *confusion matrices* to evaluate model updates.

Data visualization

Just like in the previous exercise, we use a dataset that represents different classes of objects from space.


```
import pandas
!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machine-learning
!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machine-learning
!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machine-learning

#Import the data from the .csv file
dataset = pandas.read_csv('snow_objects.csv', delimiter="\t")

# Let's have a look at the data
dataset
```

Recall that we have an *imbalanced dataset*. Some classes are much more frequent than others:

```
import graphing # custom graphing code. See our GitHub repo for details

# Plot a histogram with counts for each label
graphing.multiple_histogram(dataset, label_x="label", label_group="label", title="Label distri
```

Using binary classification

For this exercise, we'll build a *binary classification model*. We want to predict if objects in the snow are "hikers" or "not-hikers".

To do that, we first need to add another column to our dataset and set it to `True` where the original label is `hiker`, and `False` to anything else:


```
# Add a new label with true/false values to our dataset
dataset["is_hiker"] = dataset.label == "hiker"

# Plot frequency for new label
graphing.multiple_histogram(dataset, label_x="is_hiker", label_group="is_hiker", title="Distri
```

We now have only two classes of labels in our dataset, but we've made it even more imbalanced.

Let's train the random forest model using `is_hiker` as the target variable, then measure its accuracy on both *train* and *test* sets:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
# import matplotlib.pyplot as plt
# from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import accuracy_score

# Custom function that measures accuracy on different models and datasets
# We will use this in different parts of the exercise
```

```
def assess_accuracy(model, dataset, label):
    """
    Asesses model accuracy on different sets
    """
    actual = dataset[label]
    predictions = model.predict(dataset[features])
    acc = accuracy_score(actual, predictions)
    return acc

# Split the dataset in an 70/30 train/test ratio.
train, test = train_test_split(dataset, test_size=0.3, random_state=1, shuffle=True)

# define a random forest model
model = RandomForestClassifier(n_estimators=1, random_state=1, verbose=False)

# Define which features are to be used (leave color out for now)
features = ["size", "roughness", "motion"]
```

```
# Train the model using the binary label
model.fit(train[features], train.is_hiker)

print("Train accuracy:", assess_accuracy(model,train, "is_hiker"))
print("Test accuracy:", assess_accuracy(model,test, "is_hiker"))
```

Accuracy looks good for both *train* and *test* sets, but remember that this metric is not an absolute measure of success.

We should plot a confusion matrix to see how the model is actually doing:

```
# sklearn has a very convenient utility to build confusion matrices
from sklearn.metrics import confusion_matrix
import plotly.figure_factory as ff

# Calculate the model's accuracy on the TEST set
actual = test.is_hiker
predictions = model.predict(test[features])
```

```
# Build and print our confusion matrix, using the actual values and predictions
# from the test set, calculated in previous cells
cm = confusion_matrix(actual, predictions, normalize=None)

# Create the list of unique labels in the test set, to use in our plot
# I.e., ['True', 'False',]
unique_targets = sorted(list(test["is_hiker"].unique()))

# Convert values to lower case so the plot code can count the outcomes
x = y = [str(s).lower() for s in unique_targets]

# Plot the matrix above as a heatmap with annotations (values) in its cells
fig = ff.create_annotated_heatmap(cm, x, y)

# Set titles and ordering
fig.update_layout(  title_text="<b>Confusion matrix</b>",
                    yaxis = dict(categoryorder = "category descending")
                    )

fig.add_annotation(dict(font=dict(color="black",size=14),
                        x=0.5,
                        y=-0.15,
                        showarrow=False,
                        text="Predicted label",
                        xref="paper",
                        yref="paper"))

fig.add_annotation(dict(font=dict(color="black",size=14),
                        x=-0.15,
                        y=0.5,
                        showarrow=False,
                        text="Actual label",
                        textangle=-90,
                        xref="paper",
                        yref="paper"))

# We need margins so the titles fit
fig.update_layout(margin=dict(t=80, r=20, l=120, b=50))
fig['data'][0]['showscale'] = True
fig.show()
```

The confusion matrix shows us that, despite the reported metrics, the model is not incredibly precise.

Out of the 660 samples present in the *test* set (30% of the total samples), it predicted 29 *false negatives* and 33 *false positives*.

```
# Load and print umbiased set
#Import the data from the .csv file
balanced_dataset = pandas.read_csv('snow_objects_balanced.csv', delimiter="\t")

#Let's have a look at the data
graphing.multiple_histogram(balanced_dataset, label_x="label", label_group="label", title="Lab
```

This new dataset is balanced among the classes, but for our purposes we want it balanced between hikers and non-hikers.

For simplicity, let's take the hikers plus a random sampling of the non-hikers.

```
# Add a new label with true/false values to our dataset
balanced_dataset["is_hiker"] = balanced_dataset.label == "hiker"
```

```
hikers_dataset = balanced_dataset[balanced_dataset["is_hiker"] == 1]
nonhikers_dataset = balanced_dataset[balanced_dataset["is_hiker"] == False]
# take a random sampling of non-hikers the same size as the hikers subset
nonhikers_dataset = nonhikers_dataset.sample(n=len(hikers_dataset.index), random_state=1)
balanced_dataset = pandas.concat([hikers_dataset, nonhikers_dataset])

# Plot frequency for "is_hiker" labels
graphing.multiple_histogram(balanced_dataset, label_x="is_hiker", label_group="is_hiker", titl
```


You note the `is_hiker` label has the same number of `True` and `False` for both classes. We're now using a *class balanced dataset*.

```
# Test the model using a balanced dataset
actual = balanced_dataset.is_hiker
predictions = model.predict(balanced_dataset[features])

# Build and print our confusion matrix, using the actual values and predictions
# from the test set, calculated in previous cells
cm = confusion_matrix(actual, predictions, normalize=None)

# Print accuracy using this set
print("Balanced set accuracy:", assess_accuracy(model,balanced_dataset, "is_hiker"))
```

As expected, we get a noticeable drop in accuracy using a different set.

Again, let's visually analyze its performance:

```
# plot new confusion matrix
```

```
# plot new confusion matrix
# Create the list of unique labels in the test set to use in our plot
unique_targets = sorted(list(balanced_dataset["is_hiker"].unique()))

# Convert values to lower case so the plot code can count the outcomes
x = y = [str(s).lower() for s in unique_targets]

# Plot the matrix above as a heatmap with annotations (values) in its cells
fig = ff.create_annotated_heatmap(cm, x, y)

# Set titles and ordering
fig.update_layout(  title_text="<b>Confusion matrix</b>",
                    yaxis = dict(categoryorder = "category descending")
                    )

fig.add_annotation(dict(font=dict(color="black",size=14),
                        x=0.5,
                        y=-0.15,
                        showarrow=False,
                        text="Predicted label",
                        xref="paper",
                        yref="paper"))

fig.add_annotation(dict(font=dict(color="black",size=14),
                        x=-0.15,
                        y=0.5,
                        showarrow=False,
                        text="Actual label",
                        textangle=-90,
                        xref="paper",
                        yref="paper"))

# We need margins so the titles fit
fig.update_layout(margin=dict(t=80, r=20, l=120, b=50))
fig['data'][0]['showscale'] = True
fig.show()
```

The confusion matrix confirms the poor accuracy using this dataset, but why is this happening when we had such excellent metrics in

the earlier *train* and *test* sets?

Recall that the first model was heavily imbalanced. The "hiker" class made up roughly 22% of the outcomes.

When such an imbalance happens, classification models don't have enough data to learn the patterns for the minority **class**, and as a consequence become biased towards the **majority** class!

Imbalanced sets can be addressed in a number of ways:

- Improving data selection
- Resampling the dataset
- Using weighted classes

For this exercise, we'll focus on the last option.

Using class weights to balance dataset

We can assign different *weights* to the majority and minority classes, according to their distribution, and modify our training algorithm so that it takes that information into account during the training phase.

It will then penalize errors when the minority class is misclassified, in essence "forcing" the model to to better learn their features and patterns.

To use weighted classes, we have to retrain our model using the original *train* set, but this time telling the algorithm to use weights when calculating errors:

```
# Import function used in calculating weights
from sklearn.utils import class_weight

# Retrain model using class weights
# Using class_weight="balanced" tells the algorithm to automatically calculate weights for us
weighted_model = RandomForestClassifier(n_estimators=1, random_state=1, verbose=False, class_w
# Train the weighted_model using binary label
weighted_model.fit(train[features], train.is_hiker)

print("Train accuracy:", assess_accuracy(weighted_model,train, "is_hiker"))
print("Test accuracy:", assess_accuracy(weighted_model, test, "is_hiker"))
```

After using the weighted classes, the *train* accuracy remained almost the same, while the *test* accuracy showed a small improvement (roughly 1%).

Let's see if results are improved at all using the **balanced** set for predictions again:

```
print("Balanced set accuracy:", assess_accuracy(weighted_model, balanced_dataset, "is_hiker"))

# Test the weighted_model using a balanced dataset
actual = balanced_dataset.is_hiker
predictions = weighted_model.predict(balanced_dataset[features])
```

```
# Build and print our confusion matrix, using the actual values and predictions
# from the test set, calculated in previous cells
cm = confusion_matrix(actual, predictions, normalize=None)
```

The accuracy for the balanced set increased roughly 4%, but we should still try to to visualize and understand the new results.

Final confusion matrix

We can now plot a final confusion matrix, representing predictions for a *balanced dataset*, using a model trained on a *weighted class dataset*:

```
# Plot the matrix above as a heatmap with annotations (values) in its cells
fig = ff.create_annotated_heatmap(cm, x, y)
```

```
# Set titles and ordering
fig.update_layout(  title_text="<b>Confusion matrix</b>",
                    yaxis = dict(categoryorder = "category descending")
                    )

fig.add_annotation(dict(font=dict(color="black",size=14),
                        x=0.5,
                        y=-0.15,
                        showarrow=False,
                        text="Predicted label",
                        xref="paper",
                        yref="paper"))

fig.add_annotation(dict(font=dict(color="black",size=14),
                        x=-0.15,
                        y=0.5,
                        showarrow=False,
                        text="Actual label",
                        textangle=-90,
                        xref="paper",
                        yref="paper"))

# We need margins so the titles fit
fig.update_layout(margin=dict(t=80, r=20, l=120, b=50))
fig['data'][0]['showscale'] = True
fig.show()
```

While the results might look a bit disappointing, we now have 21% wrong predictions (FNs + FPs), against 25% from the previous experiment.

Correct predictions (TPs + TNs) went from 74.7% to 78.7%.

Is an all around 4% improvement significant or not?

Remember that we had relatively little data to train the model, and the features we have available may still be so similar for different samples (for example, hikers and animals tend to be small, non-rough and move a lot), that despite our efforts, the model still has some difficulty making correct predictions.

We only had to change a single line of code to get better results, so it seems worth the effort!

We only had to change a single line of code to get better results, so it seems worth the effort!

Summary

This was a long exercise, where we covered the following topics:

- Creating new label fields so we can perform *binary classification* using a dataset with multiple classes.
- How training on *imbalanced sets* can have a negative effect in perfomance, especially when using unseen data from *balanced datasets*.
- Evaluating results of *binary classification* models using a confusion matrix.
- Using weighted classes to address class imbalances when training a model and evaluating the results.

Alt + A

No compute Compute not connected Viewing Kernel not connected

Next unit: Cost functions versus evaluation metrics

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆


Alt + A