tomorrow.

Activate sandbox

Runtime File Edit View

Run all Kernel

Compute not connected

Exercise: Improving a logistic regression model

In the previous exercise, we fit a simple logistic regression model to predict the chance of an avalanche. This time, we'll improve its performance by using multiple features intelligently.

Data visualisation

Let's load our data.

```
import pandas
!pip install statsmodels
!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machine-learning/
!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machine-learning/
import graphing # custom graphing code. See our GitHub repo for details

#Import the data from the .csv file
dataset = pandas.read_csv('avalanche.csv', delimiter="\t", index_col=0)

# Split our data into training and test
import sklearn.model_selection
```

```
import sklearn.model_selection
train, test = sklearn.model_selection.train_test_split(dataset, test_size=0.25, random_state=1(

print("Train size:", train.shape[0])
print("Test size:", test.shape[0])

#Let's have a look at the data
print(train.head())
```

We have numerous features available:

- `surface_hoar` is how disturbed the surface of the snow is
- `fresh_thickness` is how thick the top layer of snow is, or 0 if there's no fresh snow on top
- `wind` is the top wind speed that day, in km/h
- `weak_layers` is the number of layers of snow that aren't well-bound to other layers
- `no_visitors` is the number of hikers who were on the trail that day
- `tracked_out` is a `1` or `0`. A `1` means that the snow has been trampled heavily by hikers

Simple logistic regression

Let's make a simple logistic regression model and assess its performance with accuracy.

```
import sklearn
from sklearn.metrics import accuracy_score
import statsmodels.formula.api as smf

# Perform logistic regression.
model = smf.logit("avalanche ~ weak_layers", train).fit()

# Calculate accuracy
def calculate_accuracy(model):
    '''
    Calculates accuracy
    '''
    # Make estimations and convert to categories
```

```
    avalanche_predicted = model.predict(test) > 0.5

    # Calculate what proportion were predicted correctly
    # We can use sklearn to calculate accuracy for us
    print("Accuracy:", accuracy_score(test.avalanche, avalanche_predicted))

calculate_accuracy(model)
```

Let's see how we can improve our model

Utilizing multiple features

Most of our features seem like they could be useful, least in theory. Let's try a model with all features we've available.

```
# Perform logistic regression.
model_all_features = smf.logit("avalanche ~ weak_layers + surface_hoar + fresh_thickness + win
calculate_accuracy(model_all_features)
```

That's a big improvement on the simpler model we've been working with.

To understand why, we can look at the summary information

```
model_all_features.summary()
```

Take a look at the P column, recalling that values less than 0.05 mean we can be confident that this parameter is helping the model make better predictions make better predictions.

Both `surface_hoar` and `wind` have very small values here, meaning they're useful predictors and probably explain why our model is working better. If we look at the `coef` (which states *parameters*) column we see that these have positive values. This means that higher winds, and greater amounts of surface hoar result in higher avalanche risk.

Simplifying our model

Looking at the summary again, we can see that `tracked_out` (how trampled the snow is), and `fresh_thickness` have large p-values. This means they aren't useful predictors. Let's see what happens if we remove them from our model:

```
# Perform logistic regression.
model_simplified = smf.logit("avalanche ~ weak_layers + surface_hoar + wind + no_visitors", tr
calculate_accuracy(model_simplified)
```

Our new model works very similarly to the old one! In some circumstances simplifying a model like this can even improve it, as it becomes less likely to overfit.

Careful feature selection

Usually, we don't just pick features blindly. Let's think about what we've just done - we removed how much fresh snow was in a

```
model_all_features.summary()
```

Let's review our earlier model again:

Look at the `fresh_thickness` row. We're told that it has a negative coefficient. This means that as thickness increases, avalanches decrease.

Similarly, `no_visitors` has a negative coefficient, meaning that fewer hikers means more avalanches.

How can this be? Well, while visitors can cause avalanches if there's a lot of fresh snow, presumably they cannot do so easily if there's no fresh snow. This means that our features aren't fully independent.

We can tell the model to try to take into account that these features interact, using a multiply sign. Let's try that now.

```
# Create a model with an interaction. Notice the end of the string where
# we've a multiply sign between no_visitors and fresh_thickness
formula = "avalanche ~ weak_layers + surface_hoar + wind + no_visitors * fresh_thickness"
model_with_interaction = smf.logit(formula, train).fit()
calculate_accuracy(model_with_interaction)
```

The model has improved to 84% accuracy! Let's look at the summary information:

```
model_with_interaction.summary()
```

We can see that the interaction term is helpful - the p-value is less than 0.05. The model is also performing better than our previous attempts.

Making predictions with multiple features

Very quickly, lets explore what this interaction means by looking at model predictions.

We will first graph two independent features in 3D. Let's start with `weak_layers` and `wind` :

```
graphing.model_to_surface_plot(model_with_interaction, ["weak_layers", "wind"], test)
```

The graph is interactive - rotate it and explore how there's a clear s-shaped relationship between the features and probability.

Let's now look at the features that we've said can interact:

```
graphing.model_to_surface_plot(model_with_interaction, ["no_visitors", "fresh_thickness"], tes
```

It looks quite different to the other! From any side, we can see an s-shape, but these combine in strange ways.

We can see that the risk goes up on days with lots of visitors *and* lots of snow. There is no real risk of avalanche when there's a lot of snow but no visitors, or when there are a lot of visitors but no snow.

The fact that it shows high risk when there's no fresh snow and no visitors could be due to rain, which keeps visitors and snow clouds away but results in avalanches of the older snow. To confirm this, we'd need to explore the data in more depth, but we'll stop here for now.

Summary

Well done! Let's recap. We've:

Well done! Let's recap. We've:

- improved our simple model by adding more features

No compute Compute not connected Viewing Kernel not connected

Next unit: Knowledge check

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆