Data Science & ML Course

Lesson #26 Deep Learning Fundamentals III

Ivanovitch Silva

December, 2018

Update from repository

```
git clone https://github.com/ivanovitchm/datascience2machinelearning.git
```

Or ….

```
git pull
```

Agenda

- Formal evaluation procedures for machine learning models
- Preparing data for deep learning
- Feature engineering
- Tackling overfitting
- The universal workflow for approaching machine learning problems
- Case study

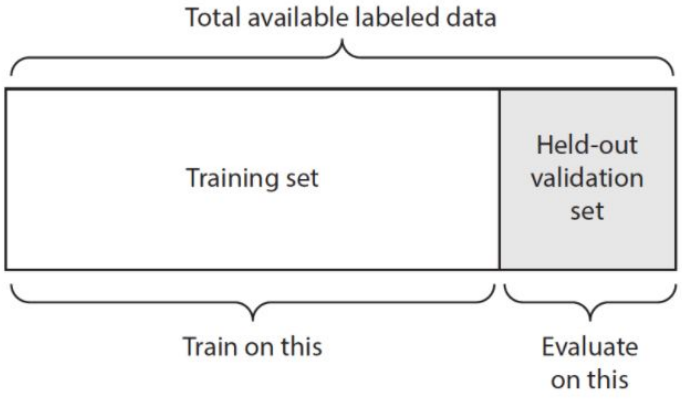
Four branches of machine learning

- Supervised learning
  - dominant form of deep learning today
- Unsupervised learning
  - dimensionality reduction and clustering
- Self-supervised learning
  - learning without annotated labels (generated by heuristic algorithm)
- Reinforcement learning
  - self-driving cars, robotics, resource management, education, and so on.

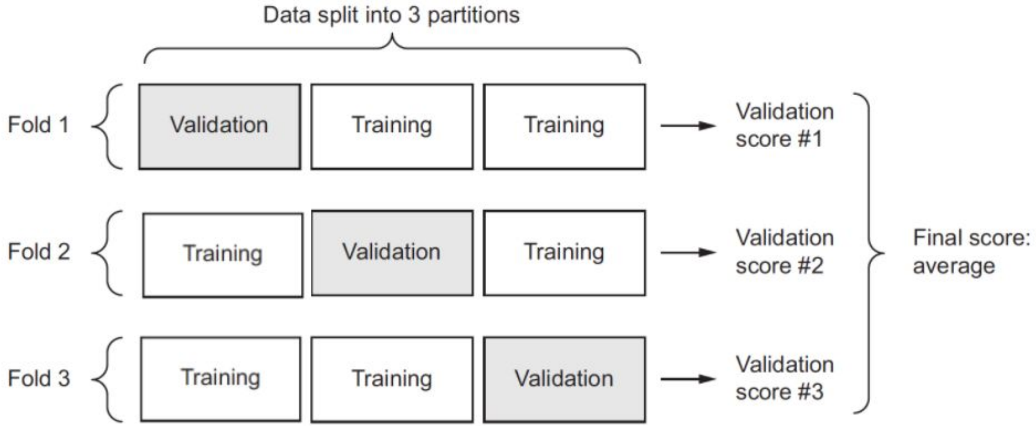
Evaluating ML models

- The goal is to achieve models that **generalize**
  - perform well on **never-before-seen data**
  - **overfitting** is the central obstacle
- Training, validation and test sets
  - Hold-out validation
  - K-Fold
  - Iterated K-Fold validation with shuffling

Evaluating ML models



Hold-Out validation



K-Fold validation

Hold-out Validation

```
# Hold-out validation pseudo-code
num_validation_samples = 10000
# Shuffling the data is usually appropriate.
np.random.shuffle(data)
# Defines the validation set
validation_data = data[:num_validation_samples]
data = data[num_validation_samples:]
# Defines the training set
training_data = data[:]
# Trains a model on the training data, and evaluates it on the validation data
model = get_model()
model.train(training_data)
validation_score = model.evaluate(validation_data)
# At this point you can tune your model, retrain it, evaluate it, tune it again...
# Once you've tuned your hyperparameters, it's common to train your final model
# from scratch on all non-test data available.
model = get_model()
model.train(np.concatenate([training_data,
validation_data]))
test_score = model.evaluate(test_data)
```

K-Fold Validation

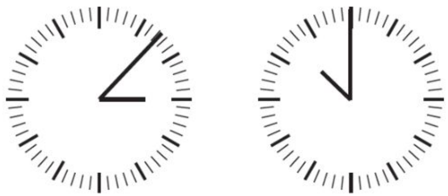
```
# K-Fold Validation
k = 4
num_validation_samples = len(data) // k
np.random.shuffle(data)
validation_scores = []
for fold in range(k):
    # Selects the validation data partition
    validation_data = data[num_validation_samples * fold:num_validation_samples * (fold + 1)]
    # Uses the remainder of the data as training data. Note that the + operator is list concatenation, not
summation
    training_data = data[:num_validation_samples * fold] + data[num_validation_samples * (fold + 1):]
    # Creates a brand-new instance of the model (untrained)
    model = get_model()
    model.train(training_data)
    validation_score = model.evaluate(validation_data)
    validation_scores.append(validation_score)
# Validation score: average of the validation scores of the k folds
validation_score = np.average(validation_scores)
# Trains the final model on all nontest data available
model = get_model()
model.train(data)
test_score = model.evaluate(test_data)
```

Data Preprocessing and Feature Engineering

Data Preprocessing

- vectorization
- normalization
- handling missing values
- feature extraction

Feature Engineering



Raw data: pixel grid		
Better features: clock hands' coordinates	{x1: 0.7, y1: 0.7} {x2: 0.5, y2: 0.0}	{x1: 0.0, y2: 1.0} {x2: -0.38, 2: 0.32}
Even better features: angles of clock hands	theta1: 45 theta2: 0	theta1: 90 theta2: 140

Underfitting and Overfitting

Underfitting and Overfitting

To prevent a model from learning misleading or irrelevant patterns found in the training data, the best solution is to **get more training data**.

- A model trained on more data will naturally generalize better.

When that isn't possible, the next-best solution:

- to modulate the quantity of information that your model is allowed to store
- to **add constraints** on what information it's allowed to store.

Regularization

- If a **network can only afford to memorize a small number of patterns**, the optimization process will force it to focus on the most prominent patterns, which have a better chance of generalizing well.
- The processing of fighting overfitting this way is called **regularization**.
  - Reducing the network size
  - Adding weight regularization (L1, L2)
  - Adding dropout

Regularization: reducing the network size

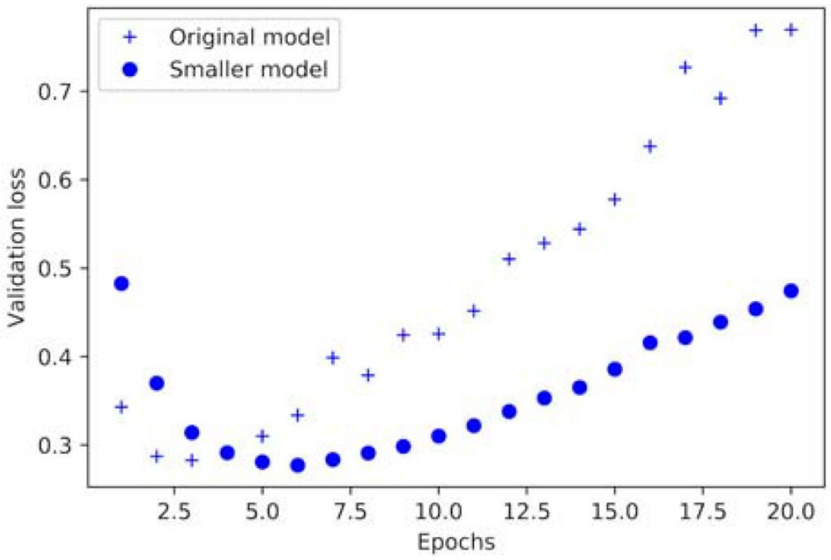
Original Size

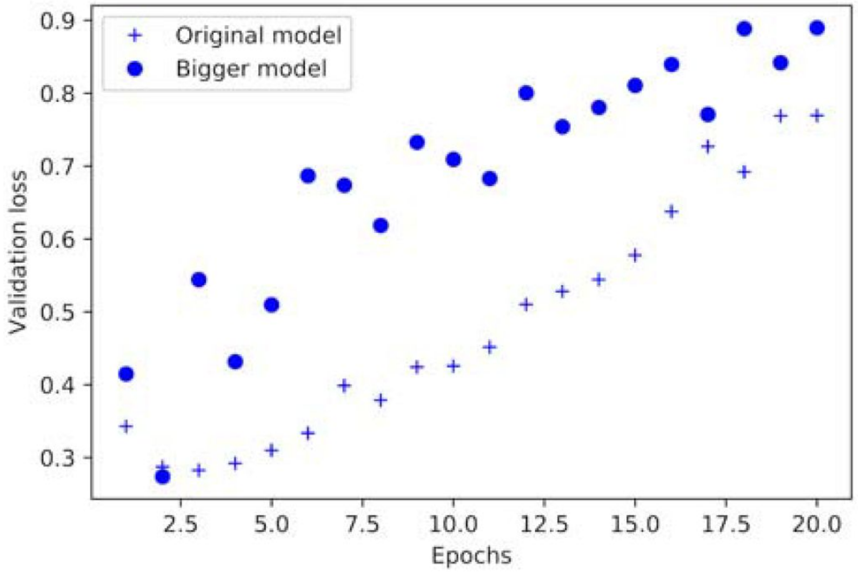
```
from keras import models
from keras import layers
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Lower capacity

```
model = models.Sequential()
model.add(layers.Dense(4, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(4, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Regularization: reducing the network size



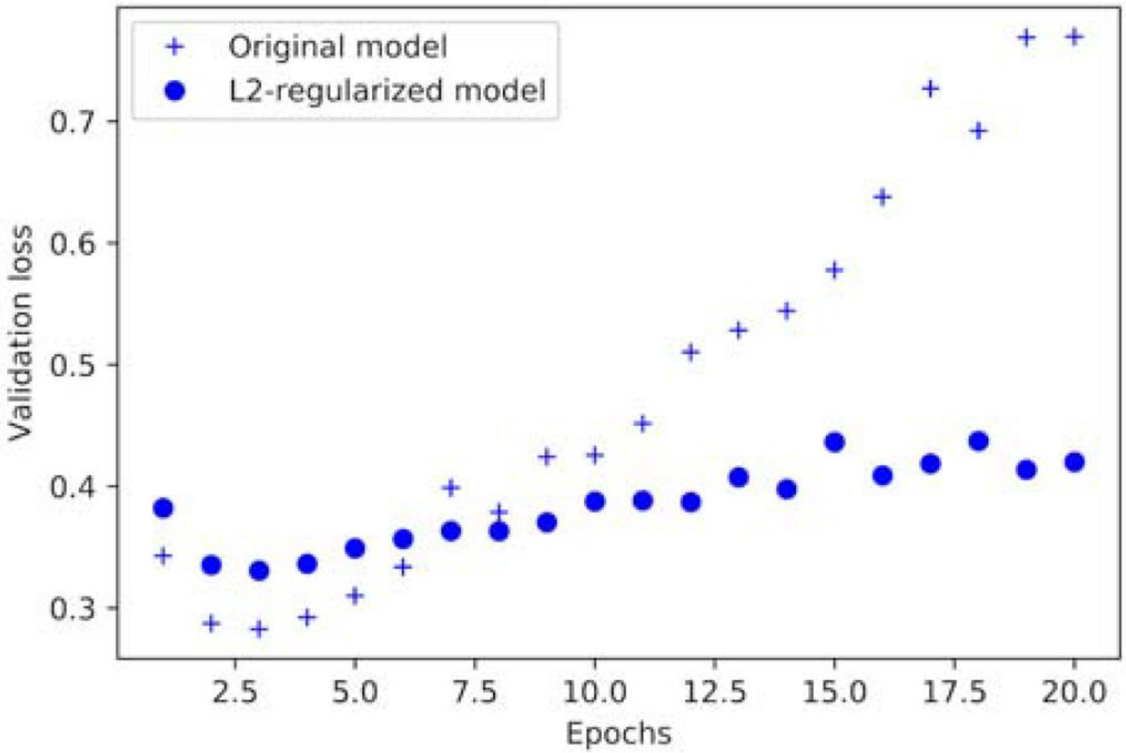


Adding weight regularization

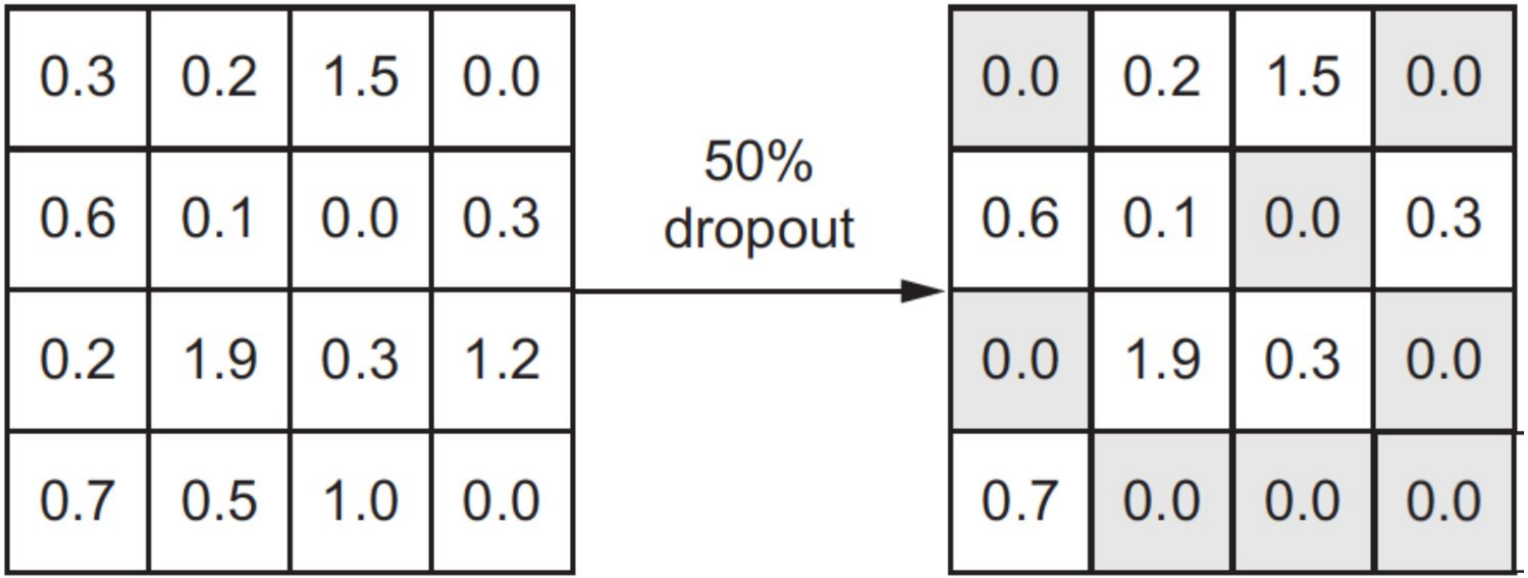
- L1 regularization
  - The cost added is proportional to the absolute value of the weight coefficients (the L1 norm of the weights).
- L2 regularization
  - The cost added is proportional to the square of the value of the weight coefficients (the L2 norm of the weights)

```
# Adding L2 weight regularization to the model
from keras import regularizers
model = models.Sequential()
model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Adding weight regularization



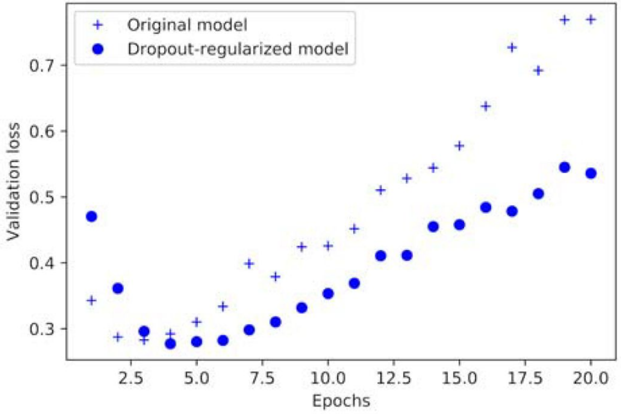
Regularization: adding dropout



The core idea is that introducing noise in the output values of a layer can break up happenstance patterns that aren’t significant

Regularization: adding dropout

```
# Adding dropout to the IMDB network
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))
```



The universal workflow of ML

1. Defining the problem and assembling a dataset
2. Choosing a measure of success
3. Deciding on an evaluation protocol
    - hold-out validation set
    - K-fold cross-validation
    - iterated K-fold validation
4. Preparing your data
    - scaled to small values (-1,1) or (0,1)
    - if different features take values in different ranges, then the data should be normalized.
    - feature engineering, especially for small-data problems.

The universal workflow of ML

5. Developing a model that does better than a baseline
   a. you need to make three key choices to build your first working model

Problem type	Last-layer activation	Loss function
Binary classification	`sigmoid`	`binary_crossentropy`
Multiclass, single-label classification	`softmax`	`categorical_crossentropy`
Multiclass, multilabel classification	`sigmoid`	`binary_crossentropy`
Regression to arbitrary values	None	`mse`
Regression to values between 0 and 1	`sigmoid`	`mse` or `binary_crossentropy`

The universal workflow of ML

6. Scaling up: developing a model that overfits
    a. add layers.
    b. make the layers bigger.
    c. train for more epochs.
    d. note: when you see that the model's performance on the validation data begins to degrade, you've achieved overfitting
7. Regularizing your model and tuning your hyperparameters
    a. Add dropout.
    b. Try different architectures: add or remove layers.
    c. Add L1 and/or L2 regularization
    d. Try different hyperparameters (such as the number of units per layer or the learning rate of the optimizer) to find the optimal configuration.
    e. Optionally, iterate on feature engineering: add new features, or remove features that don't seem to be informative.

Case Study

FIRST CLASS PASSENGERS BY AGE

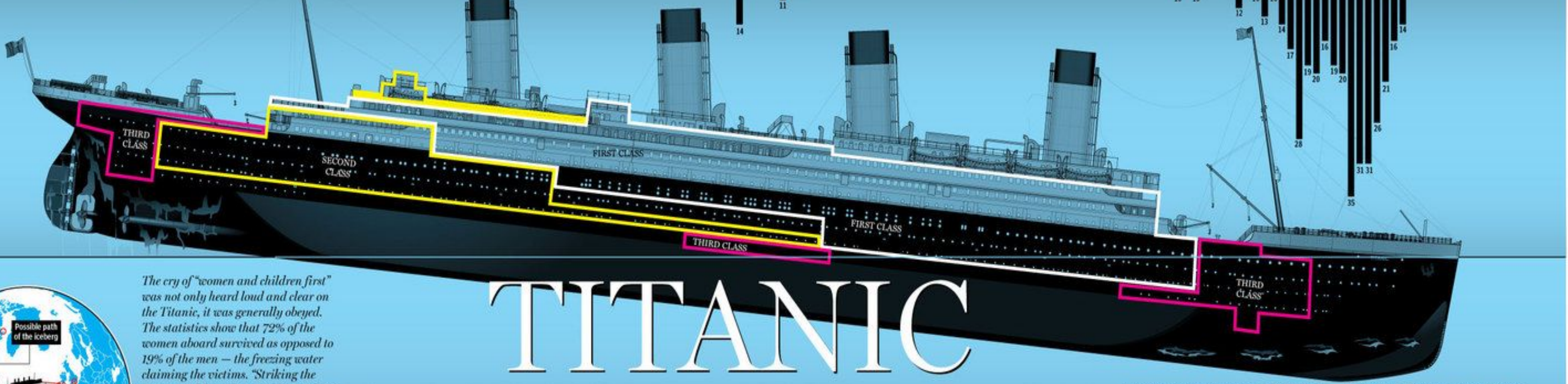
"Never have I heard such awful cries. People came tumbling down like so many oranges." — Major Arthur Peuchen, president of Standard Chemical Company and rear commodore of the Royal Canadian Yacht Club. He paid £30 10s for his ticket.

SECOND CLASS PASSENGERS BY AGE

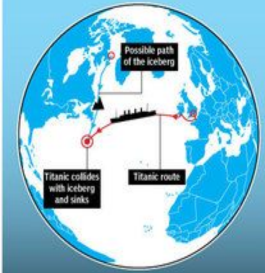
"If a ship is torpedoed, that's war. If it strikes a rock in a storm, that's nature. But just to die because there weren't enough lifeboats, that's ridiculous."
— Eva Hart, who was seven when her parents boarded the Titanic on their way to Winnipeg as Second Class passengers. Their ticket cost £26 5s.

THIRD CLASS PASSENGERS BY AGE

"With fright we heard an incredible crash and it was as if a scream from 1,000 voices came from the lit giant ship, when it broke in two and both parts rose into the sky and sank.... What was even worse than the screams were the deadly silence that came after.... It was frightful."
— Carla Andersen-Jensen, 19, on her way to the U.S. She was on a £7 17s ticket.

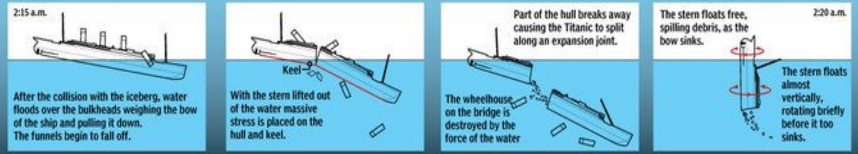


TITANIC



The cry of "women and children first" was not only heard loud and clear on the Titanic, it was generally obeyed. The statistics show that 72% of the women aboard survived as opposed to 19% of the men — the freezing water claiming the victims. "Striking the water was like a thousand knives being driven into one's body," said Second Officer Charles Lightoller. "The temperature was 28f, four degrees below freezing." But class, too, played its part in the disaster. Of the First Class passengers 62% survived, a number that dropped to 39% for Second Class passengers. Of the 700 Third Class passengers, only 24% survived.

What caused the "unsinkable" Titanic, built with the latest 20th century technology, including 16 watertight compartments, to actually go down?



DEATH RATE BY GENDER & CLASS

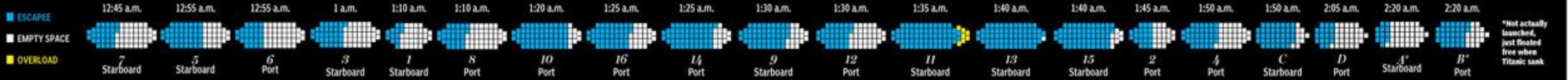
NUMBER ON BOARD	PERCENTAGE KILLED
1st class women: 141	3%
2nd class women: 92	14%
3rd class women: 179	51%
1st class men: 171	66%
2nd class men: 152	91%
3rd class men: 440	87%
1st class children: 7	14%
3rd class children: 80	69%

DEATH RATE BY NATIONALITY

NUMBER ON BOARD	PERCENTAGE KILLED
British: 327	68%
American: 306	42%
Irish: 120	65%
Swedish: 113	76%
Syrian: 81	60%
Finnish: 59	68%
Austro Hungarian: 49	84%
Canadian: 34	59%

THE LIFEBOAT LAUNCHES

The Titanic carried 20 lifeboats, enough for 1,178 people. But even though there weren't enough boats for people, many of them were still launched with empty seats. Some officers feared overfilling the lifeboats would cause them to sink.



ANDREW BARR, RICHARD JOHNSON AND GRAPHIC NEWS / NATIONAL POST

Binary Classification with Kaggle and Titanic

1. Defining the problem and assembling a dataset
   a. Kaggle gave us: train.csv, test.csv
   b. Binary classification
2. Choosing a measure of success
   a. Accuracy
3. Deciding on an evaluation protocol
   a. K-fold (dataset is small)
4. Preparing your data
   a. EDA, feature engineering

Binary Classification with Kaggle and Titanic

5. Developing a model that does better than a baseline
6. Scaling up: developing a model that overfits
7. Regularizing your model and tuning your hyperparameters

GridSearchCV

Exploring the gridsearch to create models

```
# Create different models
def create_model(id_model=0,
                 hidden=32,
                 activations="relu",
                 losses="binary_crossentropy"):
    model = models.Sequential()
    if id_model == 0:
        model.add(layers.Dense(hidden, activation=activations, input_shape=(train_data.shape[1],)))
        model.add(layers.Dense(hidden, activation=activations))
        model.add(layers.Dense(1, activation='sigmoid'))
    elif id_model == 1:
        model.add(layers.Dense(hidden, activation=activations, input_shape=(train_data.shape[1],)))
        model.add(layers.Dense(hidden, activation=activations))
        model.add(layers.Dense(hidden, activation=activations))
        model.add(layers.Dense(1, activation='sigmoid'))
        ...

    # compile the model
    model.compile(optimizer='rmsprop',loss=losses,metrics=['accuracy'])

    return model
```

Wrap Keras model to Scikit-learn

```
# Wrap keras model so it can be used by scikit-learn
model = KerasClassifier(build_fn=create_model, verbose=0)

# Create hyperparameter space
hidden_units = [64]
activations_funct = ['relu']
loss_funct = ['binary_crossentropy']
id_models = [0,8]
epochs = [50]
batch_size = [50]


# Create hpyerparameter options
hyperparameters = dict(id_model=id_models,
                       hidden=hidden_units,
                       activations=activations_funct,
                       losses=loss_funct,
                       epochs=epochs,
                       batch_size=batch_size,
                       verbose=[0])

# Create grid search
grid = GridSearchCV(estimator=model, param_grid=hyperparameters, cv=3)

# Fit grid search
grid_results = grid.fit(train_data,train_label)
```

Creating a submission file

```
submission = pd.DataFrame()
submission['PassengerId'] = df_test.PassengerId

best_model = grid_results.best_estimator_
submission['Survived'] = best_model.model.predict_classes(test_data)
# create a csv file
submission.to_csv("submission_model.csv",index=False)
```

Save and Load a Model

```
# save your model to a file
best_model.model.save("model.h5")
```

```
from keras.models import load_model

# load the model
model = load_model("model.h5")
```

The End