Machine Learning

Walt Disney
ANIMATION STUDIOS

RAJESH SHARMA
Walt Disney Animation Studios

Thank you to ACM SIGGRAPH!

Pol Jeremias-Vila : SIGGRAPH 2021 Chair

Tomasz Bednarz: Frontiers Program Chair

Alex Bryant: Student Volunteers Chair

Tim Hendrickson: Digital Marketing Manager

Student Volunteers:

Rogelio, Trinity, Aurora, Emily, Hunter & Kendra

Machine Learning

Rajesh Sharma

Bindu Reddy

CEO, Abacus.ai

- **CEO and Founder**
  Abacus.ai
- **General Manager - AI and Deep Learning**
  Amazon Web Services (AWS)
- **CEO and Co-Founder**
  Post Intelligence
- **Head of Product - Google Apps**
  Google

Today

- Quick Recap: Intro & Housing data
- First Neural Network - Regression
- Visualization
- Second Neural Network - Classification

Recap

Given some data:

Q: How do we approximate a function that represents the data?

A: By minimizing prediction error

Recap - Terminology

- *Given: Features (X, Attributes), Output (Y, Labels, Ground Truth): Y=f(X)*
- *Network (Model)*
- *Loss Function (Metric, Cost)*
- *Activation Function (adds non-linearity, Ex: sigmoid, ReLU)*
- *Training (fit, Optimization to minimize Loss Function)*
- *Evaluate (performance, correctness)*
- *Predict (Inference, on new data)*

Hands-on

- ★ Log in to your google drive
- ★ Make a shortcut to: `https://bit.ly/3oKCVCh`
- ★ Make a copy of:
    - HousingRegression.ipynb,
    - FlowerClassification.ipynb
    - *Let's take a look at the HousingRegression.ipynb*
        - *Simple Neural Network for predicting home prices*

How

- FRAMING: What is observed & what answer you want to predict
- DATA COLLECTION: Collect, clean, and prepare data
- DATA ANALYSIS: Visualize & analyze the data
- FEATURE PROCESSING: Transform raw data for better predictive input
- MODEL BUILDING: Design and build the learning algorithm
- TRAINING: Feed data to the model and evaluate the quality of the models
- PREDICTION: Use model to generate predictions for new data instances

ML project - process - apply to housing problem 1/7

- *FRAMING*: what is observed & what answer you want to predict

Observed: parameters related to homes for a census block

Predict: Median home-price for the block

Housing project steps - 2/7

- *DATA COLLECTION*: Collect, clean, and prepare data

Already given in a csv file: `housing.csv`

```
isna(), np.where(), dropna()
```

Housing project steps - 3/7

- *DATA ANALYSIS*: Visualize & analyze the data
    - `sns.pairplot(...)`
    - `data.describe()`

Housing project steps - 4/7

- *FEATURE PROCESSING*: Transform raw data for better predictive input

-- Normalize `(x-x.min())/(x.max()-x.min())` : brings data between 0 and 1

-- Standardize `(x-x.mean())/x.std()`: remaps to mean of 0, and std_dev of 1

-- Keep 20% for testing:

```
train=data.sample(frac=0.8)

test=data.drop(train.index)
```

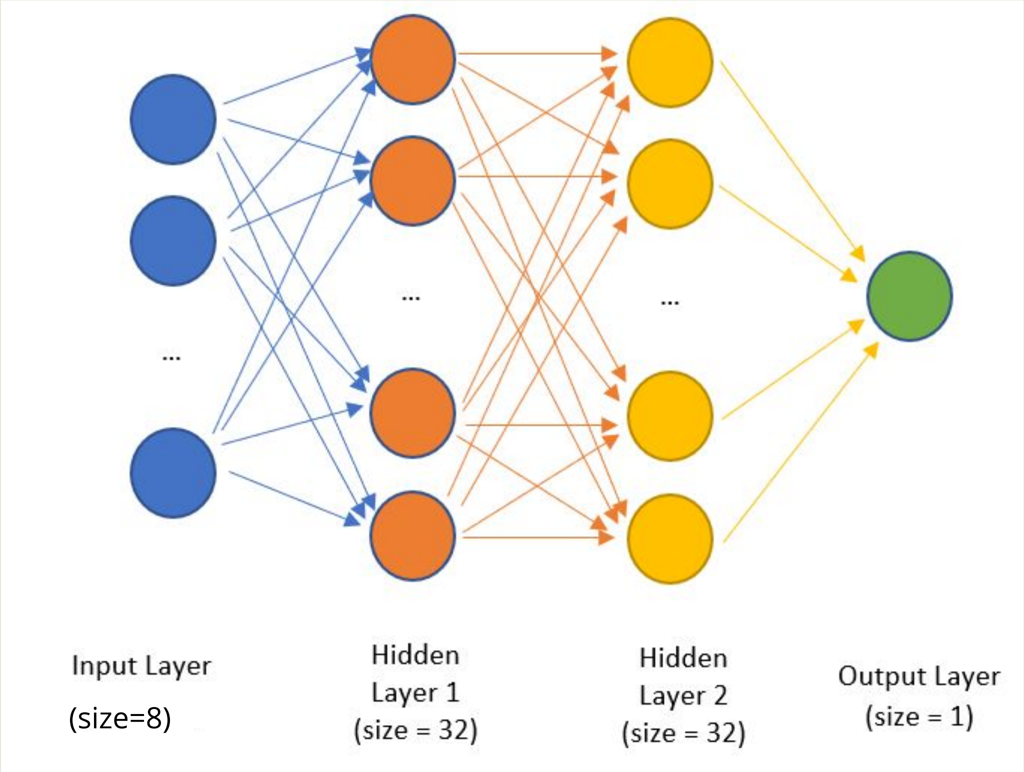
-- Separate features (x) from labels (y):

```
X_train = train.drop('median_house_value', axis=1)

Y_train = train[‘median_house_value’]
```

Housing project steps - 5/7

- *MODEL BUILDING*: Feed features to learning algorithm to build models



Housing project steps - 5/7

- *MODEL BUILDING*: Feed features to learning algorithm to build models

```
import tensorflow as tf

INPUT_SHAPE=[9]

model = tf.keras.Sequential([
    tf.keras.layers.InputLayer(INPUT_SHAPE, name="Input_Layer"),
    tf.keras.layers.Dense(32, activation='relu', name="dense_01"),
    tf.keras.layers.Dense(32, activation='relu', name="dense_02"),
    tf.keras.layers.Dense(1, name="Output_Layer")
  ])

model.compile(loss='mse',
              optimizer=tf.keras.optimizers.RMSprop(0.001),
              metrics=['mae', 'mse'])

print(model.summary())
```

Housing project steps - 6/7

- *TRAINING*: compute weights and Evaluate the quality of the models

```
example_batch = x_train[:10]
example_result = model.predict(example_batch)
print(example_result)

history = model.fit(x_train, y_train,
                    batch_size=32,
                    epochs=10,
                    validation_split=0.2,
                    verbose=1)

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validate'], loc='upper left')
plt.show()

loss, mae, mse = model.evaluate(x_test, y_test, verbose=2)
print("Loss:", loss, " mae:", mae, " mse:", mse)
```

Housing project steps - 7/7

- *PREDICTION*: Use model to generate predictions for new data instances

```
p_test = model.predict(x_test)
print(p_test, y_test)

a = plt.axes(aspect='equal')
plt.scatter(y_test, p_test)
plt.xlabel('True Values')
plt.ylabel('Predictions')
lims = [0, 1]
plt.xlim(lims)
plt.ylim(lims)
plt.plot(lims, lims)
plt.show()

error = p_test.flatten() - y_test
print(error)
plt.hist(error, bins = 25)
plt.xlabel("Prediction Error")
plt.ylabel("Count")
plt.show()
```

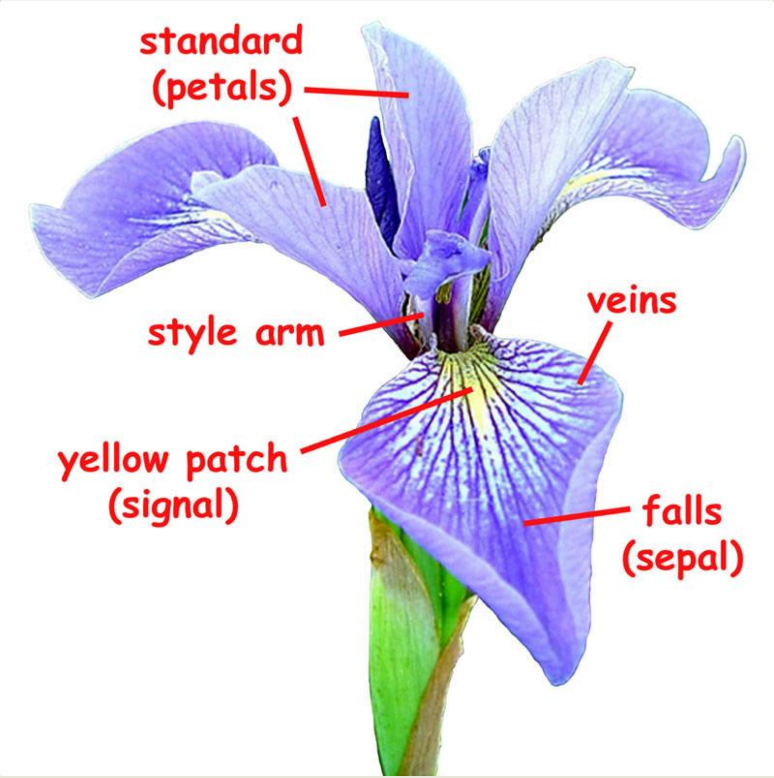
Housing project steps - Homework

- *Improve results*, tune hyperparameters
  - Increase/Decrease # of epochs
  - Try different batch sizes: 16 (observe time to train, accuracy etc)
  - Try different learning rates (0.01, 0.000001)
  - Try different optimizers ('adam')
  - Try different loss functions ('mse', 'mae')
  - Make the network deeper (more layers) or denser (more nodes/layer)

Flower Project - Classification

- *FRAMING*: what is observed & what answer you want to predict
  - Given data: about 3 species of iris flowers
- *DATA COLLECTION*: Collect, clean, and prepare data
- *DATA ANALYSIS*: Visualize & analyze the data
- *FEATURE PROCESSING*: Transform raw data for better predictive input:
- *MODEL BUILDING*: Feed features to learning algorithm to build models
- *TRAINING*: Evaluate the quality of the models
- *PREDICTION*: Use model to generate predictions for new data instances

Flower Project - Classification



Flower Classification: get data ready

	120	4	setosa	versicolor	virginica
0	6.4	2.8	5.6	2.2	2
1	5.0	2.3	3.3	1.0	1
2	4.9	2.5	4.5	1.7	2
3	4.9	3.1	1.5	0.1	0
4	5.7	3.8	1.7	0.3	0

Given: Features about Iris: sepal length, sepal width, petal length, petal width
Task: Classify into kind of Iris: setosa (0), versicolor (1), or virginica (2)

- Plot data
- Split data for training and testing
- Normalize training data

Flowers - Classification - get data ready

```
    #----------DATA READING
filename = 'https://storage.googleapis.com/download.tensorflow.org/data/iris_training.csv'
# read file
csv_data = pd.read_csv(filename, sep=',')
print(csv_data.head())

column_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
class_names = ['Iris setosa', 'Iris versicolor', 'Iris virginica']

#----------DATA CLEANUP
csv_data.columns = column_names # new_header --set the header row as the data header
print(csv_data.head())
# look at simple data statistics
print(csv_data.describe().transpose())
# plot of all features against each other
sns.pairplot(csv_data)
```

Flowers - Classification - get data ready

```
#----------TRAIN/TEST SPLIT
train_data = csv_data.sample(frac=0.8) # take 80% randomly from the data for training
test_data = csv_data.drop(train_data.index) # reserve the rest for testing

# separate out the y (results) from x (features) for training
x_train = train_data.drop('species', axis=1)
y_train = train_data['species']
# normalize the training data
x_train = (x_train-x_train.min())/(x_train.max()-x_train.min())

# separate out the y (results) from x (features) testing
x_test = test_data.drop('species', axis=1)
y_test = test_data['species']
# normalize the test data
x_test = (x_test-x_test.min())/(x_test.max()-x_test.min())

print('Training Data\n', x_train.describe().transpose())
print('Test Data\n', x_test.describe().transpose())
```

Flowers - Classification steps - model

```
#--------MODEL BUILDING
num_params = len(x_train.keys())
print(num_params)
model = tf.keras.Sequential([
   tf.keras.layers.InputLayer([num_params], name="Input_Layer"),
   tf.keras.layers.Dense(32, activation='relu', name="dense_01"),
   tf.keras.layers.Dense(32, activation='relu', name="dense_02"),
   # 1 node in the output for the median_house_vale
   tf.keras.layers.Dense(3, name="Output_Layer")
 ])

model.compile(optimizer=tf.keras.optimizers.RMSprop(0.001),
             # loss function to minimize
             loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
             # list of metrics to monitor
             metrics=['acc',])

model.summary()
```

Flowers - Classification Log Likelihood

`Note:`

`Loss Function: SparseCategoricalCrossentropy(from_logits=True)`

- Instead of a value, it returns a 'LOG LIKELIHOOD' for each output class
- Which we convert into a probability for each output class
- We take the class with the highest probability as the predicted class

Log Likelihood:

```
    class-A      class-B     class-C
 [ 0.02669345  0.03092438 -0.01683718 ]
```

Convert to probabilites (using *softmax* function)

```
 [ 0.13765042   0.739082   0.12326758 ]
```

Output class: B (class with the maximum probability)

Flowers - Classification - train and test

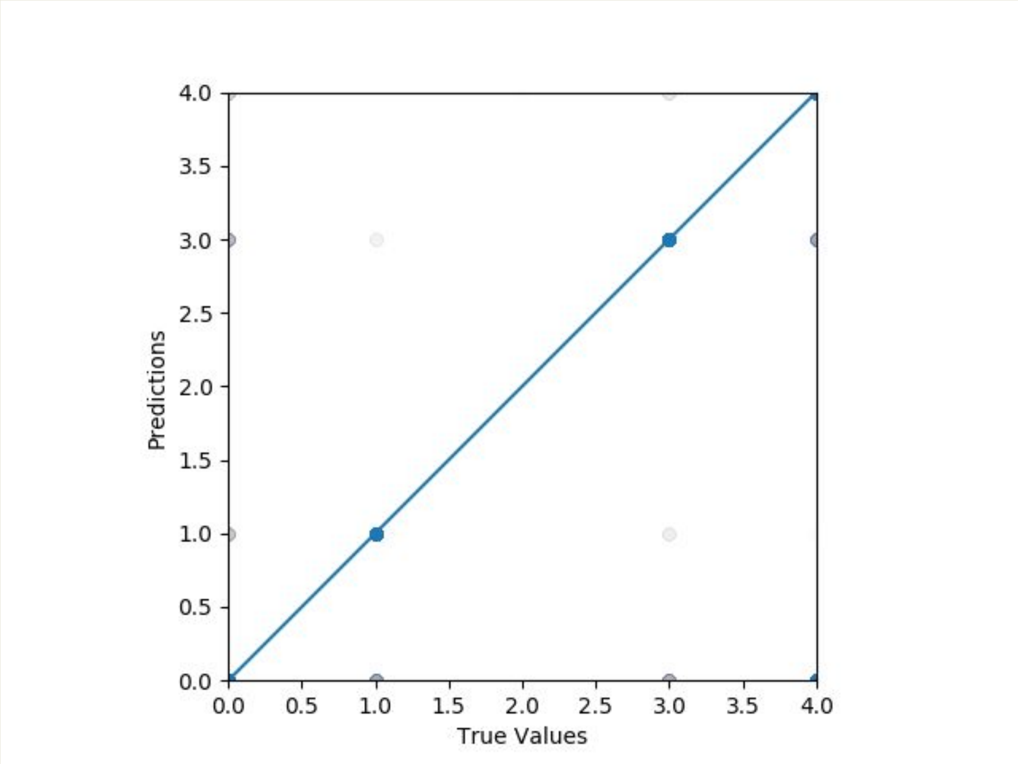
```
# Fit/TRAIN model on training data
history = model.fit(x_train, y_train,
                    batch_size=4,
                    epochs=10,
                    validation_split=0.2,
                    verbose=1)
#--------MONITOR
# Plot training & validation loss values
fig = plt.figure(figsize=(12,9))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validate'], loc='upper left')
plt.show()
```

Classification - Evaluation

- Problem with Prediction vs True Value
- Truth Table
- Confusion Matrix

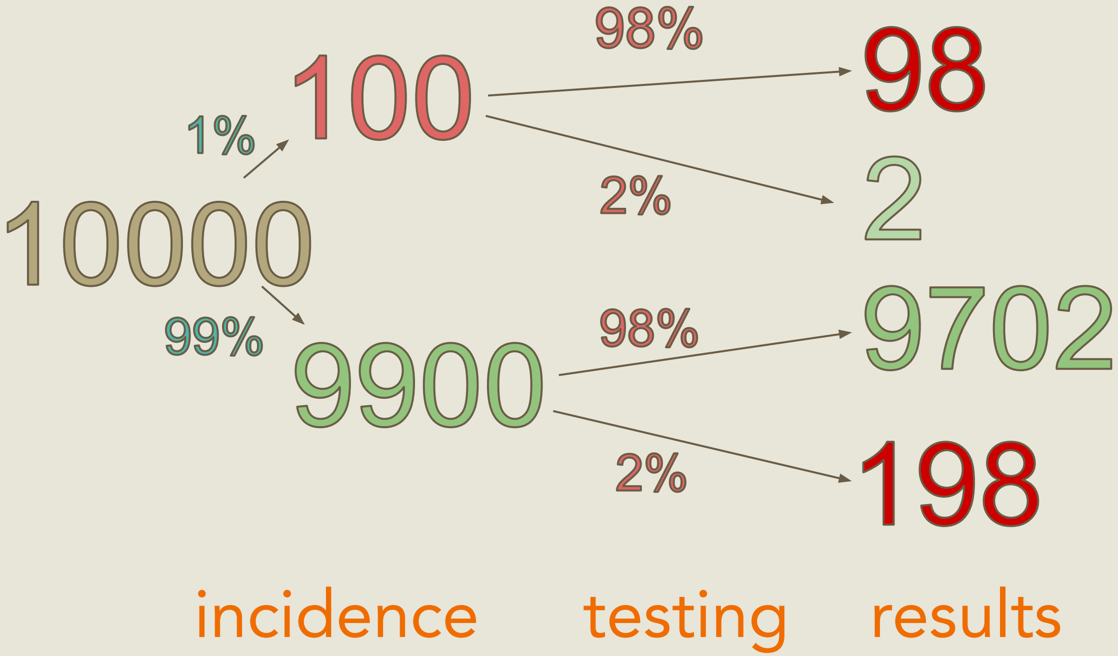
Classification - Evaluation

- Problem with Prediction vs True Value



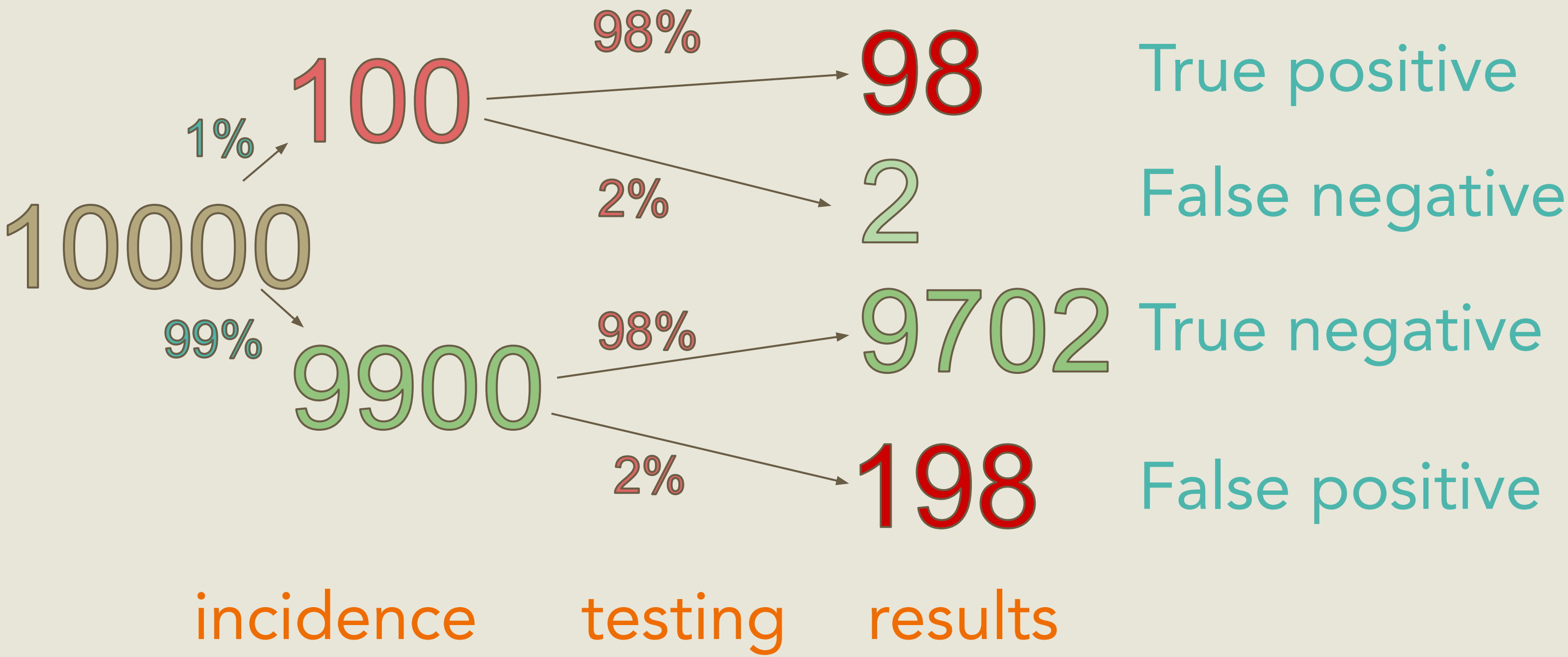
Classification - Evaluation

- Fallacy of accurate tests:
  - Ex: 1% incidence, test is: 98% accurate



Classification - Evaluation

- Fallacy of accurate tests:
  - Ex: 1% incidence, test is: 98% accurate



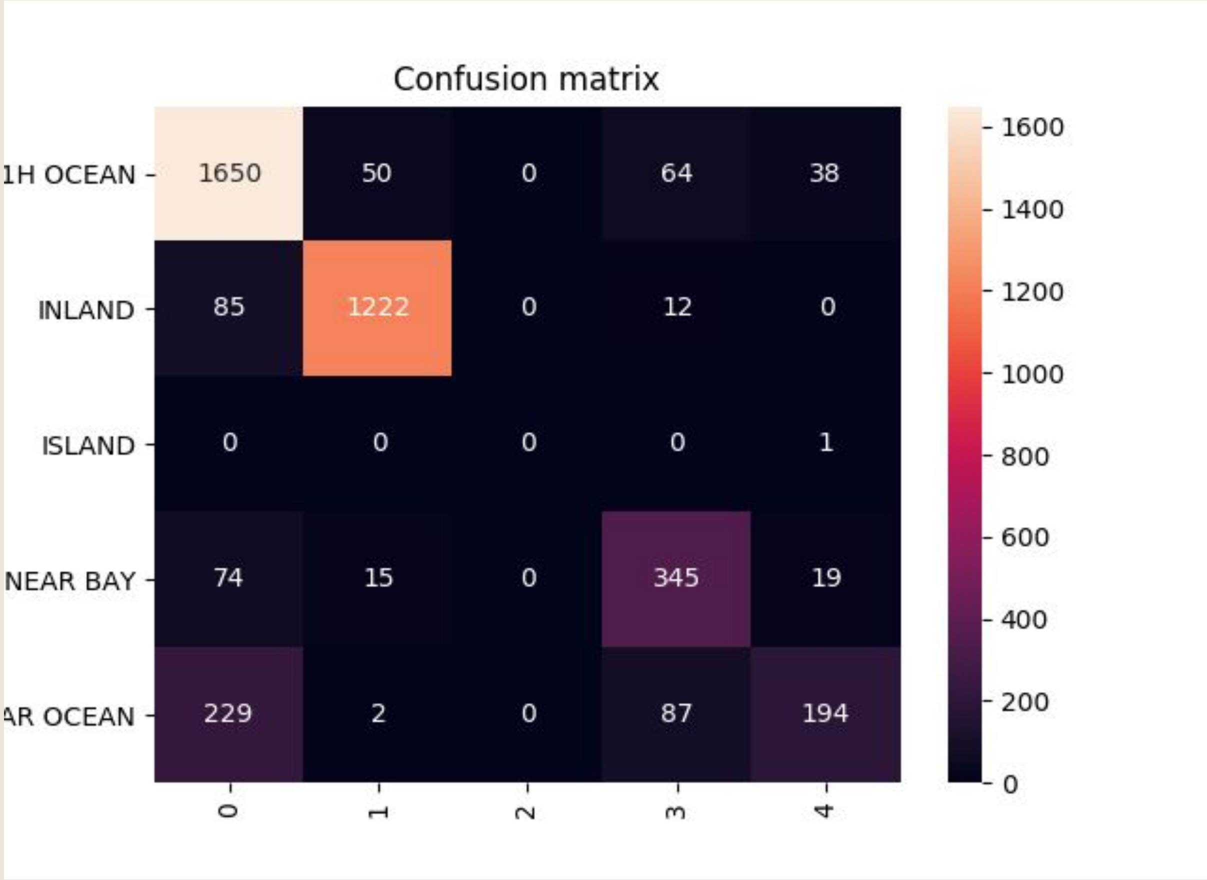
Classification - Evaluation - Confusion Matrix

Actual Value

Predicted Value

	positives	negatives
positives	TRUE POSITIVE	FALSE POSITIVE
negatives	FALSE NEGATIVE	TRUE NEGATIVE

Classification - Evaluation - Confusion Matrix*



Classification - Evaluation - Confusion Matrix

```
# plot the confusion matrix as heatmap
sns.heatmap(tf.math.confusion_matrix(y_test,
p_test_class), cmap="Blues", annot=True)
```

Summary

- Regression, Classification
- Optimizer, Loss Function
- Model, training and prediction
- Visualization of progress, results

Next Class

- Data Compression using AutoEncoder
- Tensorflow Data Pipeline
- Homework:
    - Try different hyperparameters
    - Extra credit: Do a regression for sin(x)
- @xarmalarma, #siggraph2021