A Guide to Turi Create
Task focused ML for your apps
Session 712

Aaron Franklin, Turi
Zach Nation, Turi
Create Core ML models for your intelligent apps.
What Do These Apps Have in Common?
What Do These Apps Have in Common?

Use machine learning
What Do These **Apps** Have in Common?

- Use machine learning
- Require very little data
What Do These Apps Have in Common?

- Use machine learning
- Require very little data
- Model created with Turi Create, deployed with Core ML
What Do These Apps Have in Common?

Use machine learning

Require very little data

Model created with Turi Create, deployed with Core ML

Follow a 5 step recipe
What Do These Apps Have in Common?

- Use machine learning
- Require very little data
- Model created with Turi Create, deployed with Core ML
- Follow a 5 step recipe
- Demo apps in the labs
What is Turi Create?
What is Turi Create?

Python library for creating Core ML models
What is Turi Create?

Python library for creating Core ML models

Easy to use, no need to be an ML expert
What is Turi Create?

Python library for creating Core ML models
Easy to use, no need to be an ML expert
Task focused APIs
What is Turi Create?

Python library for creating Core ML models

Easy to use, no need to be an ML expert

Task focused APIs

Cross platform (Mac, Linux)
What is Turi Create?

Python library for creating Core ML models

Easy to use, no need to be an ML expert

Task focused APIs

Cross platform (Mac, Linux)

Open source
Get the newest beta release
Get the newest beta release

> pip install turicreate==5.0b1
5 step recipe for creating Core ML models
Task

What are you trying to do?
Task

What are you trying to do?

Data

What data do you need?
Task
What are you trying to do?

Data
What data do you need?

Model
Create a model
Task: What are you trying to do?

Data: What data do you need?

Model: Create a model

Evaluate: Is the model any good?
Task
What are you trying to do?

Data
What data do you need?

Model
Create a model

Evaluate
Is the model any good?

Deploy
Add Core ML model to your app
Step 1: Task

- Task
- Data
- Model
- Evaluate
- Deploy
<table>
<thead>
<tr>
<th>I want to...</th>
<th>Machine Learning Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label Images</td>
<td>Image Classification</td>
</tr>
<tr>
<td>Recognize objects within images</td>
<td>Object Detection</td>
</tr>
<tr>
<td>Find similar images</td>
<td>Image Similarity</td>
</tr>
<tr>
<td>Create stylized avatars / profile images</td>
<td>Style Transfer</td>
</tr>
<tr>
<td>Personalize choices for users</td>
<td>Recommender</td>
</tr>
<tr>
<td>Detect an activity using sensors</td>
<td>Activity Classification</td>
</tr>
<tr>
<td>Analyze sentiment of messages</td>
<td>Text Classifier</td>
</tr>
<tr>
<td>Predict a label</td>
<td>Classifiers</td>
</tr>
<tr>
<td>Predict numeric values</td>
<td>Regression</td>
</tr>
<tr>
<td>Group similar datapoints together</td>
<td>Clustering</td>
</tr>
</tbody>
</table>
Image Classification
Classify images into categories

Rose
Object Detection
Detect objects in images
Object Detection
Detect objects in images
Style Transfer
Apply a visual style to images
Style Transfer
Apply a visual style to images
Activity Classification
Recognize gestures or motion
Activity Classification
Recognize gestures or motion
Recommenders
Personalize user preferences
import turicreate

// Load data
data = turicreate.SFrame("data.sframe")
train, test = data.random_split(0.8)

// Create a model
model = turicreate.object_detector.create(train, "labels")

// Evaluate the model
metrics = model.evaluate(test)

// Export for deployment
model.export_coreml("MyModel.mlmodel")
import turicreate

// Load data
data = turicreate.SFrame("data.sframe")
train, test = data.random_split(0.8)

// Create a model
model = turicreate.object_detector.create(train, "labels")

// Evaluate the model
metrics = model.evaluate(test)

// Export for deployment
model.export_coreml("MyModel.mlmodel")
import turicreate

data = turicreate.SFrame("data.sframe")
train, test = data.random_split(0.8)

model = turicreate.object_detector.create(train, "labels")

metrics = model.evaluate(test)

model.export_coreml("MyModel.mlmodel")
import turicreate

// Load data
data = turicreate.SFrame("data.sframe")
train, test = data.random_split(0.8)

// Create a model
model = turicreate.object_detector.create(train, "labels")

// Evaluate the model
metrics = model.evaluate(test)

// Export for deployment
model.export_coreml("MyModel.mlmodel")
import turicreate

// Load data
data = turicreate.SFrame("data.sframe")
train, test = data.random_split(0.8)

// Create a model
model = turicreate.object_detector.create(train, "labels")

// Evaluate the model
metrics = model.evaluate(test)

// Export for deployment
model.export_coreml("MyModel.mlmodel")
import turicreate

// Load data
data = turicreate.SFrame("data.sframe")
train, test = data.random_split(0.8)

// Create a model
model = turicreate.object_detector.create(train, "labels")

// Evaluate the model
metrics = model.evaluate(test)

// Export for deployment
model.export_coreml("MyModel.mlmodel")
import turicreate

// Load data
data = turicreate.SFrame("data.sframe")
train, test = data.random_split(0.8)

// Create a model
model = turicreate.object_detector.create(train, "labels")

// Evaluate the model
metrics = model.evaluate(test)

// Export for deployment
model.export_coreml("MyModel.mlmodel")
import turicreate

// Load data
data = turicreate.SFrame("classifier_data.sframe")
train, test = data.random_split(0.8)

// Create a model
model = turicreate.image_classifier.create(train, "labels")

// Evaluate the model
metrics = model.evaluate(test)

// Export for deployment
model.export_coreml("MyModel.mlmodel")
import turicreate

data = turicreate.SFrame("sensor_data.sframe")
train, test = data.random_split(0.8)

model = turicreate.activity_classifier.create(train, "labels")

metrics = model.evaluate(test)

model.export_coreml("MyModel.mlmodel")
Sample App: Object Detection

Scenario: Calorie counting

Identify:
• Foods
• Where they occur
Sample App: Object Detection

Scenario: Calorie counting

Identify:
• Foods
• Where they occur
Step 2: Data
[{
  "label" : "coffee",
  "coordinates" : {
      "x" : 387,
      "y" : 660,
      "height" : 550,
      "width" : 814
  }
},
{
  "label" : "croissant",
  "coordinates" : {
      "x" : 800,
      "y" : 630,
      "height" : 373,
      "width" : 812
  }
}]
### SFrame

**Tabular data structure**

<table>
<thead>
<tr>
<th>image</th>
<th>annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Image 1" /></td>
<td></td>
</tr>
</tbody>
</table>
```
{
    "label": "coffee",
    "coordinates": {
        "x": 387,
        "y": 660,
        "height": 550,
        "width": 814,
    }
}
```
|
| ![Image 2](image) | 
```
{
    "label": "waffle",
    "coordinates": {
        "x": 261,
        "y": 420,
        "height": 500,
        "width": 800,
    }
}
```
|
| ![Image 3](image) | 
```
{
    "label": "egg",
    "coordinates": {
        "x": 381,
        "y": 419,
        "height": 431,
        "width": 700,
    }
}
```
What is an SFrame
What is an SFrame

Disk backed, tabular data structure
What is an SFrame

- Disk backed, tabular data structure
- Common data manipulation tasks
What is an SFrame

- Disk backed, tabular data structure
- Common data manipulation tasks
- Work with text, images, and json
What is an SFrame

- Disk backed, tabular data structure
- Common data manipulation tasks
- Work with text, images, and json
- Interactively explore and visualize data
import turicreate

// Load annotations & images
annotations = turicreate.SFrame("annotations.csv")
images = turicreate.load_images("training_images")

// Interactive visualization & exploration
images.explore()

// Row-wise access
first_row = annotations[0]

// Column wise access
label_column = annotations["labels"]

// Common operations
data = images.join(annotations)
data.save("data.sframe")
import turicreate

// Load annotations & images
annotations = turicreate.SFrame("annotations.csv")
images = turicreate.load_images("training_images")

// Interactive visualization & exploration
images.explore()

// Row-wise access
first_row = annotations[0]

// Column wise access
label_column = annotations["labels"]

// Common operations
data = images.join(annotations)
data.save("data.sframe")
import turicreate

// Load annotations & images
annotations = turicreate.SFrame("annotations.csv")
images = turicreate.load_images("training_images")

// Interactive visualization & exploration
images.explore()

// Row-wise access
first_row = annotations[0]

// Column wise access
label_column = annotations["labels"]

// Common operations
data = images.join(annotations)
data.save("data.sframe")
import turicreate

// Load annotations & images
annotations = turicreate.SFrame("annotations.csv")
images = turicreate.load_images("training_images")

// Interactive visualization & exploration
images.explore()

// Row-wise access
first_row = annotations[0]

// Column wise access
label_column = annotations["labels"]

// Common operations
data = images.join(annotations)
data.save("data.sframe")
```python
import turicreate

// Load annotations & images
annotations = turicreate.SFrame("annotations.csv")
images = turicreate.load_images("training_images")

// Interactive visualization & exploration
images.explore()

// Row-wise access
first_row = annotations[0]

// Column wise access
label_column = annotations["labels"]

// Common operations
data = images.join(annotations)
data.save("data.sframe")
```
import turicreate

// Load annotations & images
annotations = turicreate.SFrame("annotations.csv")
images = turicreate.load_images("training_images")

// Interactive visualization & exploration
images.explore()

// Row-wise access
first_row = annotations[0]

// Column wise access
label_column = annotations["labels"]

// Common operations
data = images.join(annotations)
data.save("data.sframe")
import turicreate

// Load annotations & images
annotations = turicreate.SFrame("annotations.csv")
images = turicreate.load_images("training_images")

// Interactive visualization & exploration
images.explore()

// Row-wise access
first_row = annotations[0]

// Column wise access
label_column = annotations["labels"]

// Common operations
data = images.join(annotations)
data.save("data.sframe")
Step 3: Model

- Task
- Data
- Model
- Evaluate
- Deploy
Model Creation Simplified

// Create a model
model = turicreate.object_detector.create(train, "annotations")

- Model creation customized to task
- State of the art
- Small or large amounts of data
Step 4: Evaluate
```python
import turicreate

data = turicreate.SFrame("data.sframe")
train, test = data.random_split(0.8)

model = turicreate.object_detector.create(train, "annotations")

metrics = model.evaluate(test)

model.export_coreml("MyModel.mlmodel")
```
Evaluating Object Detectors

Did you get the label right?
Did you get the bounding box right?
Evaluation Metric

Need **correct labels** AND **at least 50%** overlap in boxes
Evaluation Metric

Need correct labels AND at least 50% overlap in boxes

Overlap 10%

❌

Need **correct labels** AND **at least 50%** overlap in boxes
Evaluation Metric

Need **correct labels** AND **at least 50%** overlap in boxes
Evaluation Metric

Need **correct labels** AND **at least 50%** overlap in boxes
Step 5: Deployment
import turicreate

// Load data
data = turicreate.SFrame("data.sframe")
train, test = data.random_split(0.8)

// Create a model
model = turicreate.object_detector.create(train, "annotations")

// Evaluate the model
metrics = model.evaluate(test)

// Export for deployment
model.export_coreml("MyModel.mlmodel")
```swift
let mlModel = try MLModel(contentsOf: modelURL)
let visionModel = try VNCoreMLModel(for: mlModel)
let objectRecognition = VNCoreMLRequest(model: visionModel,
    completionHandler: { (request, error) in
        guard let results = request.results else { return }
        for case let foundObject as VNRecognizedObjectObservation in results {
            let bestLabel = foundObject.labels.first! // Label with highest confidence
            let objectBounds = foundObject.boundingBox
            // Use the computed values.
            print(bestLabel.identifier, bestLabel.confidence, objectBounds)
        }
    }
})
```

| Vision with Core ML | Executive Ballroom | Thursday 3:00PM |
import turicreate

data = turicreate.SFrame("data.sframe")
train, test = data.random_split(0.8)

model = turicreate.object_detector.create(train, "annotations")

metrics = model.evaluate(test)

model.export_coreml("MyModel.mlmodel")
Demo

Calorie counting

Zach Nation, Turi
Demo Recap

Loaded images and annotations into the SFrame
Interactively explored data
Created a model
Evaluated quantitatively
Exported to Core ML
Exciting New features in 5.0
Turi Create 5.0
Turi Create 5.0

New Task
• Style Transfer
Turi Create 5.0

New Task
• Style Transfer

Performance
• Native GPU Acceleration on Macs
Turi Create 5.0

New Task
• Style Transfer

Performance
• Native GPU Acceleration on Macs

New Deployments
• Recommenders
• Vision Feature Print powered models
Style Transfer
Style Transfer
Style Transfer
Style Transfer
Style Transfer
import turicreate

// Load data
content_images = turicreate.load_images("content/")
style_images = turicreate.load_images("style/")

// Create a model
model = turicreate.style_transfer.create(content_images, style_images)

// Make predictions
stylized_images = model.stylize(content_images)

// Export for deployment
model.export_coreml("MyStyles.mlmodel")
Demo
Stylish filter creation app
Demo Recap

- Loaded images into the SFrame
- Created a model
- Stylized images
- Visualized predictions
- Exported to Core ML
Mac GPU acceleration

12x Image classifier

9x Object detector

Metal for Accelerating Machine Learning

Hall 3

Thursday 4:00PM
Personalization

Task: Recommend items for users

Data: Historical preferences

Deployment:
• Core ML custom models
• macOS 10.14, iOS 12

Top community feature request
Machine Learning Model
Name: MyRecommender
Type: Custom Model
Size: 3.4 MB
Author: Johnny Applased
Description: Item Similarity Recommender Model exported from Turi Create 5.0x2
License: MIT

Model Class
MyRecommender
Model is not part of any target. Add the model to a target to enable generation of the model class.

Model Evaluation Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>interactions</td>
<td>Dictionary (Int64 -&gt; Double)</td>
<td>User interactions where keys are item IDs and values are sentinel values.</td>
</tr>
<tr>
<td>k</td>
<td>Int64</td>
<td>Return the top k recommendations.</td>
</tr>
<tr>
<td>Outputs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>recommendations</td>
<td>Dictionary (Int64 -&gt; Double)</td>
<td>Top k recommendations.</td>
</tr>
<tr>
<td>probabilities</td>
<td>Dictionary (Int64 -&gt; Double)</td>
<td>The probability for each recommendation in the top k.</td>
</tr>
</tbody>
</table>

Dependencies

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom Models</td>
<td></td>
</tr>
<tr>
<td>TCR recommender</td>
<td>Turi Create Recommender support for Core ML</td>
</tr>
</tbody>
</table>
Machine Learning Model

- Name: MyRecommender
- Type: Custom Model
- Size: 3.4 MB
- Author: Johnny Applassed
- Description: Item Similarity Recommender Model exported from Turi Create 5.0x2
- License: MIT

Model Class

- MyRecommender

Model is not part of any target. Add the model to a target to enable generation of the model class.

Model Evaluation Parameters

- Inputs
  - interactions: Dictionary (Int64 + Double)
  - k: Int64
  - Description: User interactions where keys are item IDs and values are sentinel values.
  - Return the top k recommendations.

- Outputs
  - recommendations: Dictionary (Int64 + Double)
  - probabilities: Dictionary (Int64 + Double)
  - Description: Top k recommendations.
  - The probability for each recommendation in the top k.

Dependencies

- Custom Models
  - TCReducer: Turi Create Recommender support for Core ML.
let model = MyRecommender()

// Historical interactions
let input = [
    "BrownBeard": 1.0,
    "BlackHandleBar": 1.0,
    "BrownLongHair": 1.0
]

// Make predictions
let output = try model.prediction(interactions: input, k: 10)

// Consume predictions
let predictions = output.recommendations
let confidence  = output.probabilities
Recap
Create Core ML models for your intelligent apps

Task

Data

Model

Evaluate

Deploy
import turicreate

// Load data
data = turicreate.SFrame("data.sframe")
train, test = data.random_split(0.8)

// Create a model
model = turicreate.object_detector.create(train, "annotations")

// Evaluate the model
metrics = model.evaluate(test)

// Export for deployment
model.export_coreml("MyModel.mlmodel")