Data Science

with

Images and Text

Materials at https://github.com/joshloyal/pydata-amazon-products

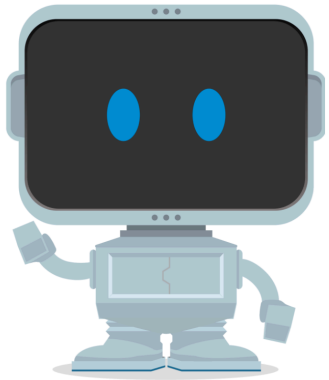
DataRobot

Introduction

Joshua Loyal: Data Scientist at DataRobot
- joshua@datarobot.com

Igor Veksler: Data Scientist at DataRobot
- igor.veksler@datarobot.com

Code for this talk can be found here
- https://github.com/joshloyal/pydata-amazon-products



Welcome to ML-Mart

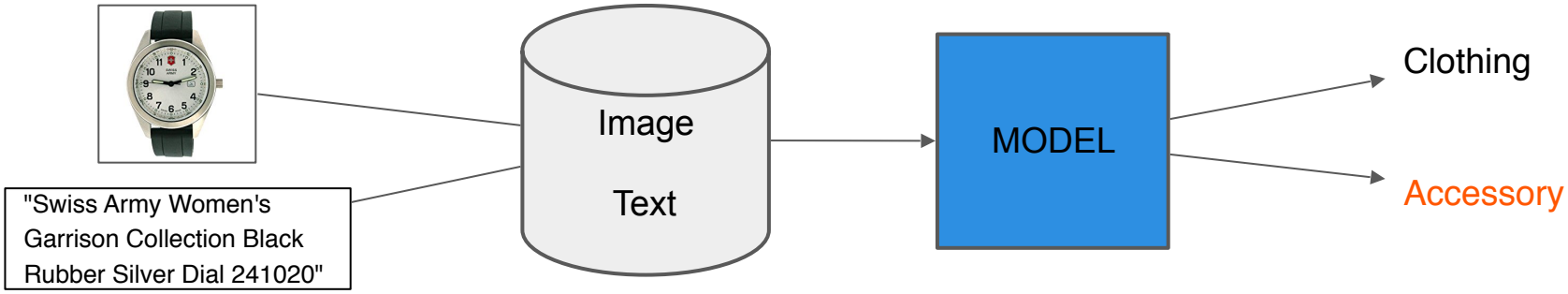
Product Classification

Business Problem: A piece of merchandise is added to our store's inventory with a picture and description from the manufacturer. We would like to automatically tag these items for display in our online store.

Welcome to ML-Mart

Product Classification

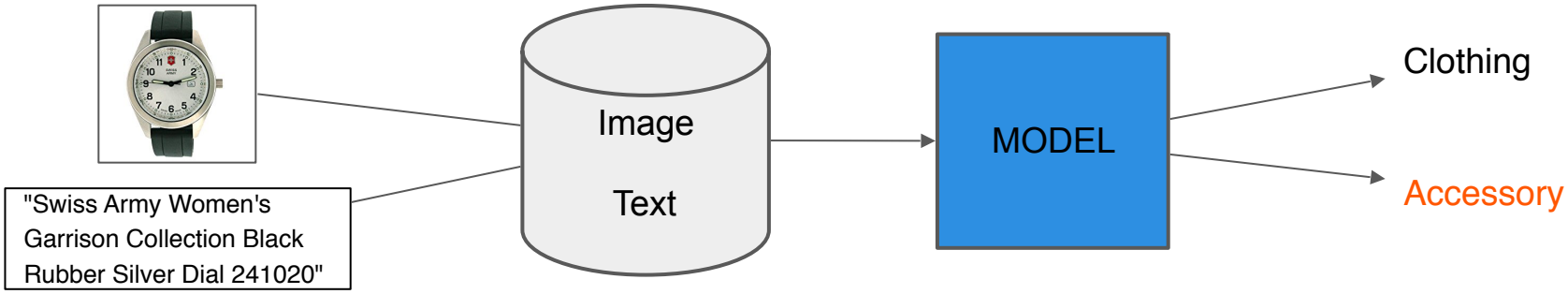
Business Problem: A piece of merchandise is added to our store's inventory with a picture and description from the manufacturer. We would like to automatically tag these items for display in our online store.



Welcome to ML-Mart

Product Classification

Business Problem: A piece of merchandise is added to our store's inventory with a picture and description from the manufacturer. We would like to automatically tag these items for display in our online store.



Data Science Problem:

What is the probability a product is either Clothing or an Accessory?

Meet the Data!

Image-based recommendations on styles and substitutes
J. McAuley, C. Targett, J. Shi, A. van den Hengel
SIGIR, 2015

Inferring networks of substitutable and complementary products
J. McAuley, R. Pandey, J. Leskovec
Knowledge Discovery and Data Mining, 2015

Meet the Data!

Image-based recommendations on styles and substitutes
J. McAuley, C. Targett, J. Shi, A. van den Hengel
SIGIR, 2015

Inferring networks of substitutable and complementary products
J. McAuley, R. Pandey, J. Leskovec
Knowledge Discovery and Data Mining, 2015

```
amazon_products.csv
|—— images/
|   |—— B00FDP6M9A.jpg
|   |—— B0077DSN7U.jpg
|   |—— B0050SMHQW.jpg
|   |—— B007XF1WHK.jpg
|            ...
```

Meet the Data!

Image-based recommendations on styles and substitutes
J. McAuley, C. Targett, J. Shi, A. van den Hengel
SIGIR, 2015

Inferring networks of substitutable and complementary products
J. McAuley, R. Pandey, J. Leskovec
Knowledge Discovery and Data Mining, 2015

```
amazon_products.csv
|—— images/
|   |—— B00FDP6M9A.jpg
|   |—— B0077DSN7U.jpg
|   |—— B0050SMHQW.jpg
|   |—— B007XF1WHK.jpg
|            ...
```

```python
import pandas as pd

df = pd.read_csv('amazon_products.csv')
df.head(10)
```

Meet the Data!

Image-based recommendations on styles and substitutes
J. McAuley, C. Targett, J. Shi, A. van den Hengel
SIGIR, 2015

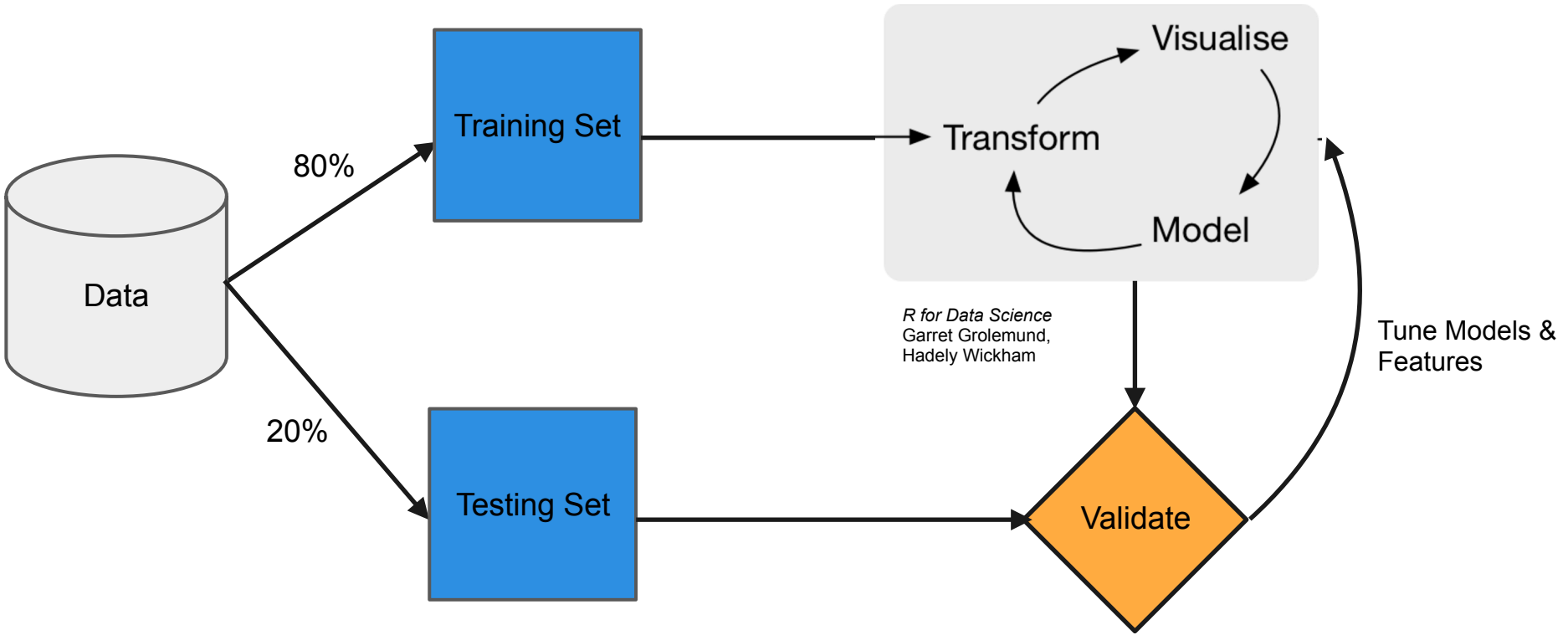
Inferring networks of substitutable and complementary products
J. McAuley, R. Pandey, J. Leskovec
Knowledge Discovery and Data Mining, 2015

```
amazon_products.csv
|—— images/
|   |—— B00FDP6M9A.jpg
|   |—— B0077DSN7U.jpg
|   |—— B0050SMHQW.jpg
|   |—— B007XF1WHK.jpg
|          ...
```

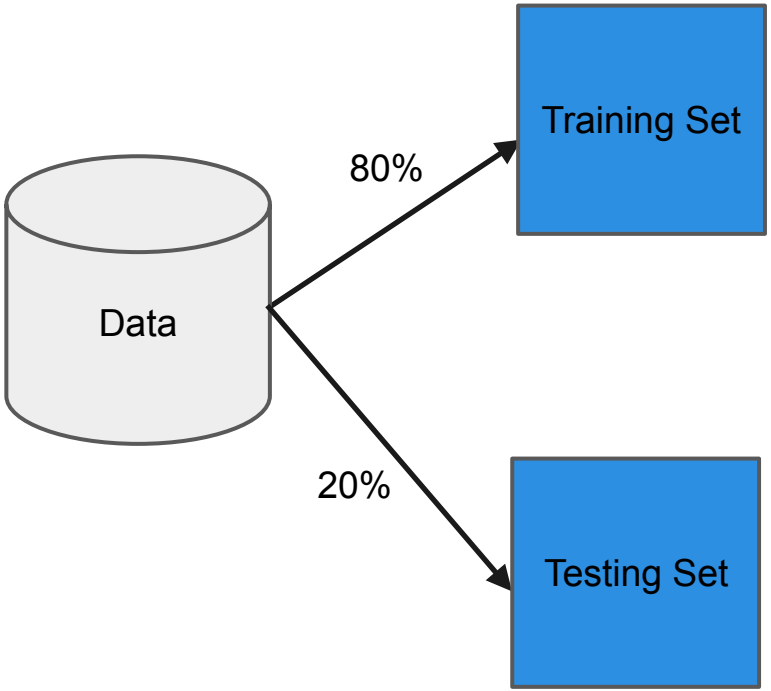
```
import pandas as pd

df = pd.read_csv('amazon_products.csv')
df.head(10)
```

	title	image_file	product_category
0	Coffee Retro Women Lady Weave Rivet Leather St...	B00FDP6M9A.jpg	Accessory
1	Fabric Fanny Pack- High Quality- Color Pattern...	B0077DSN7U.jpg	Clothing
2	Missoni Women's SM12 Ballet Flat,Beige,37.5 EU...	B0050SMHQW.jpg	Accessory
3	Brown Reindeer Hat Chenille X Small	B007XF1WHK.jpg	Clothing
4	ADJUSTABLE MULTI COLOR Cross Howlite Turquoise...	B008TUQY1C.jpg	Accessory
5	2 Pieces of Pink Pearl Beaded Head Chain	B00AKJISCS.jpg	Accessory
6	Lemon Beads	B004KV3JR6.jpg	Clothing
7	Sterling Silver Celtic Knot Ring	B003TPHD1M.jpg	Accessory
8	Quiksilver Men's All Time Long Sleeve Surf T-S...	B00EKR9WD0.jpg	Clothing
9	Red Tree Design Quartz Clock Pendant Pocket Watch	B007RK16W2.jpg	Accessory

Modeling Workflow



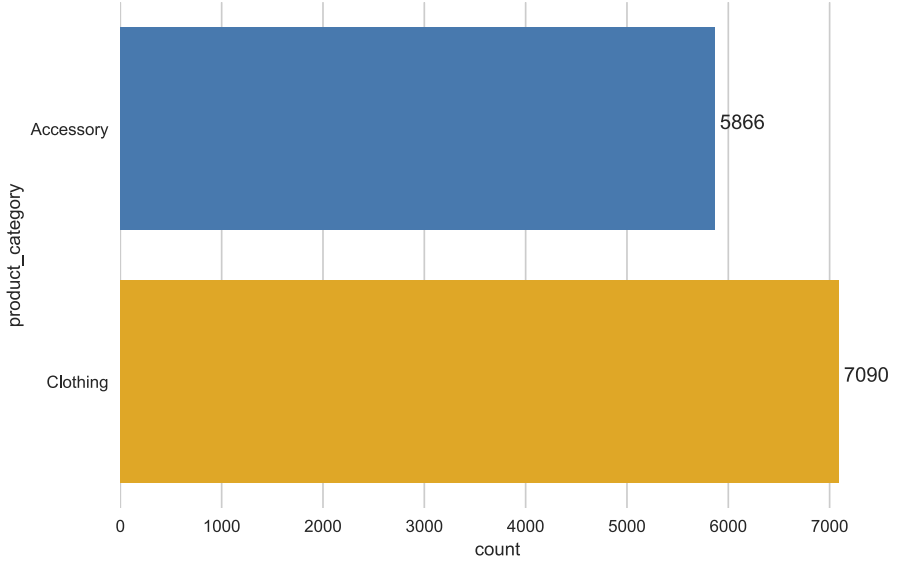
Modeling Workflow



How should we split the data?

Look at the Target!

```
import seaborn as sns

colors = ["windows blue", "amber"]

with sns.xkcd_palette(colors):
    ax = sns.countplot(y='product_category',
                       data=df)

    # display counts directly on graph
    for patch in ax.patches:
        x = patch.get_bbox().get_points()[1, 0]
        y = patch.get_bbox().get_points()[:, 1]

        offset = 50
        ax.annotate('{:d}'.format(
            int(x)), (x + offset, np.mean(y)))

    sns.despine(left=True, bottom=True)
```



Stratified Split

Convert Pandas DataFrames to NumPy arrays.

```
from sklearn.model_selection import train_test_split

column_names = df.columns
y = df.pop('product_category').values
X = df.values

# train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=
    stratify=y,
    random_state=

# save train/test splits to a file
train_df = pd.DataFrame(
    np.hstack((X_train, y_train[:, np.newaxis])),
    columns=column_names)

test_df = pd.DataFrame(
    np.hstack((X_test, y_test[:, np.newaxis])),
    columns=column_names)

train_df.to_csv(
                index=
test_df.to_csv(
               index=
```

Stratified Split

Dataset is imbalanced.
Use a stratified split with a fix random state!

```
from sklearn.model_selection

column_names = df.columns
y = df.pop(
X = df.values

# train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    stratify=y,
    random_state=123)

# save train/test splits to a file
train_df = pd.DataFrame(
    np.hstack((X_train, y_train[:, np.newaxis])),
    columns=column_names)

test_df = pd.DataFrame(
    np.hstack((X_test, y_test[:, np.newaxis])),
    columns=column_names)

train_df.to_csv(
                index=
test_df.to_csv(
               index=
```

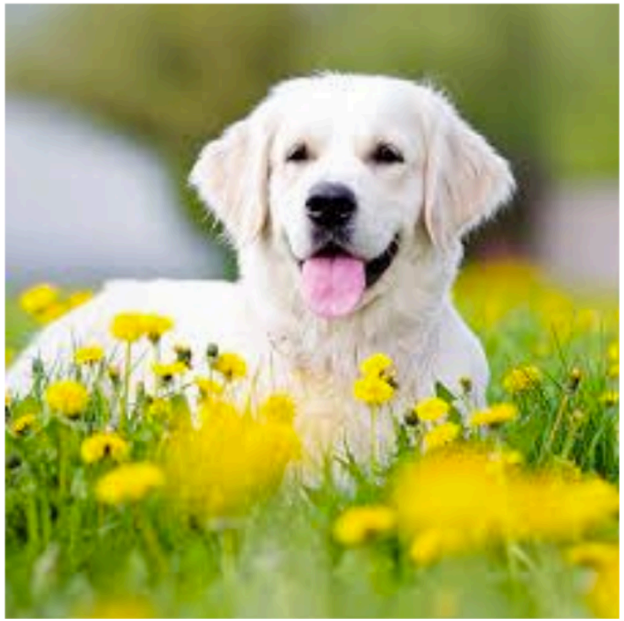
Stratified Split

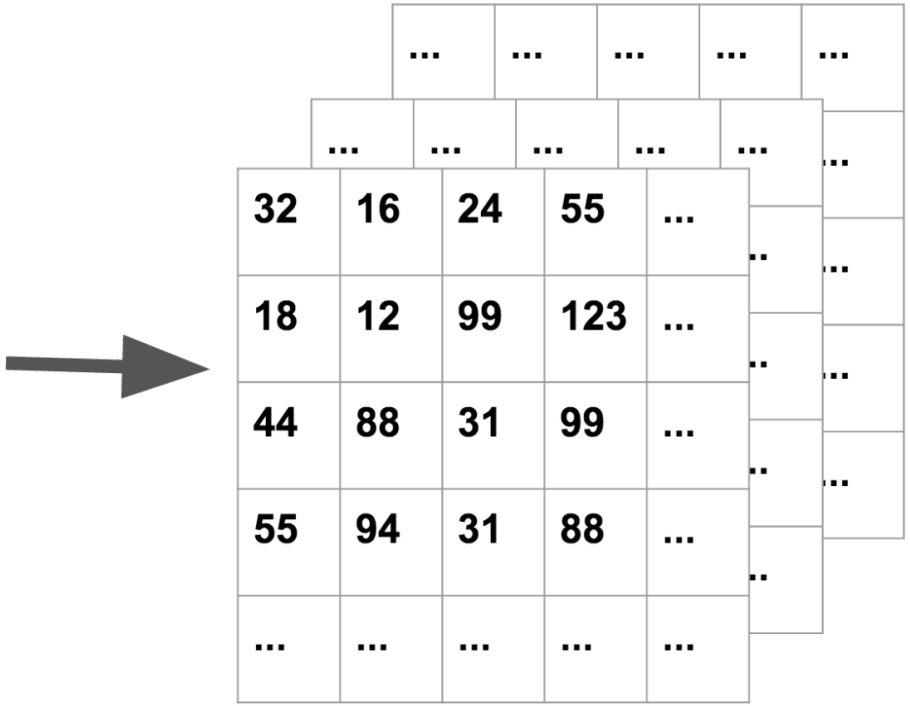
Save to disk for use across experiments.

```
from sklearn.model_selection

column_names = df.columns
y = df.pop(
X = df.values

# train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=
    stratify=y,
    random_state=

# save train/test splits to a file
train_df = pd.DataFrame(
    np.hstack((X_train, y_train[:, np.newaxis])),
    columns=column_names)

test_df = pd.DataFrame(
    np.hstack((X_test, y_test[:, np.newaxis])),
    columns=column_names)

train_df.to_csv('amazon_products_train.csv',
                index=False)
test_df.to_csv('amazon_products_test.csv',
               index=False)
```

Images

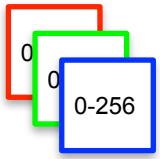
How are Images Represented?



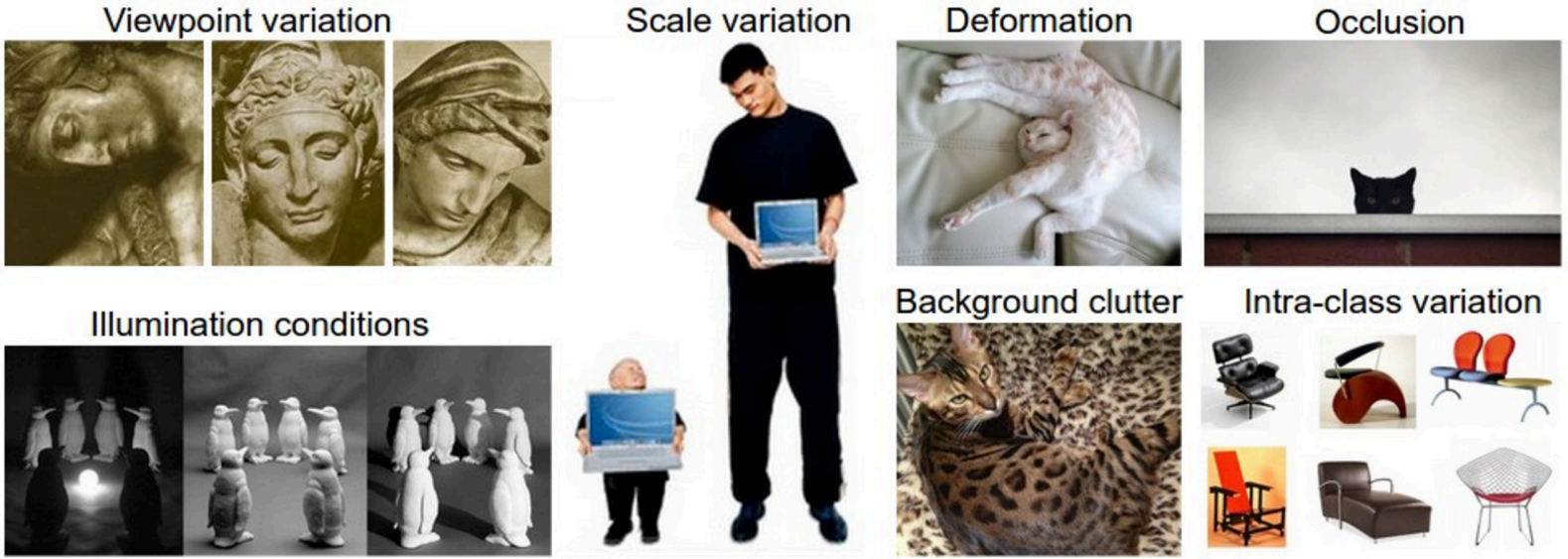


Loading Images in Python

```
import PIL.Image as pil_image

# load image with pillow
image_file = 'path/to/image.jpg'
img = pil_image.open(image_file).convert('RGB')

# re-size this to 224 x 224 (width x height) pixels
target_size = (224, 224)
img = img.resize(target_size, pil_image.LANZCOS)

# convert to a numpy array
img = np.asarray(img, np.uint8)
```

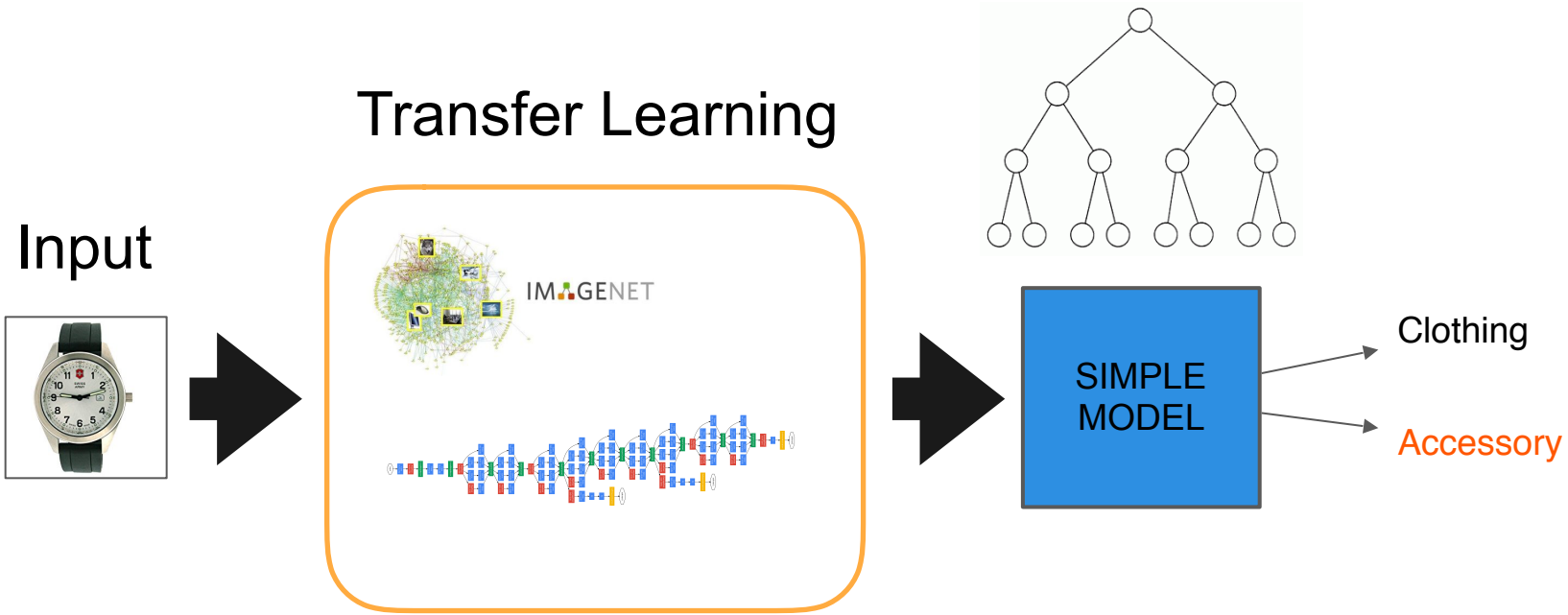


Challenges of Image Classification

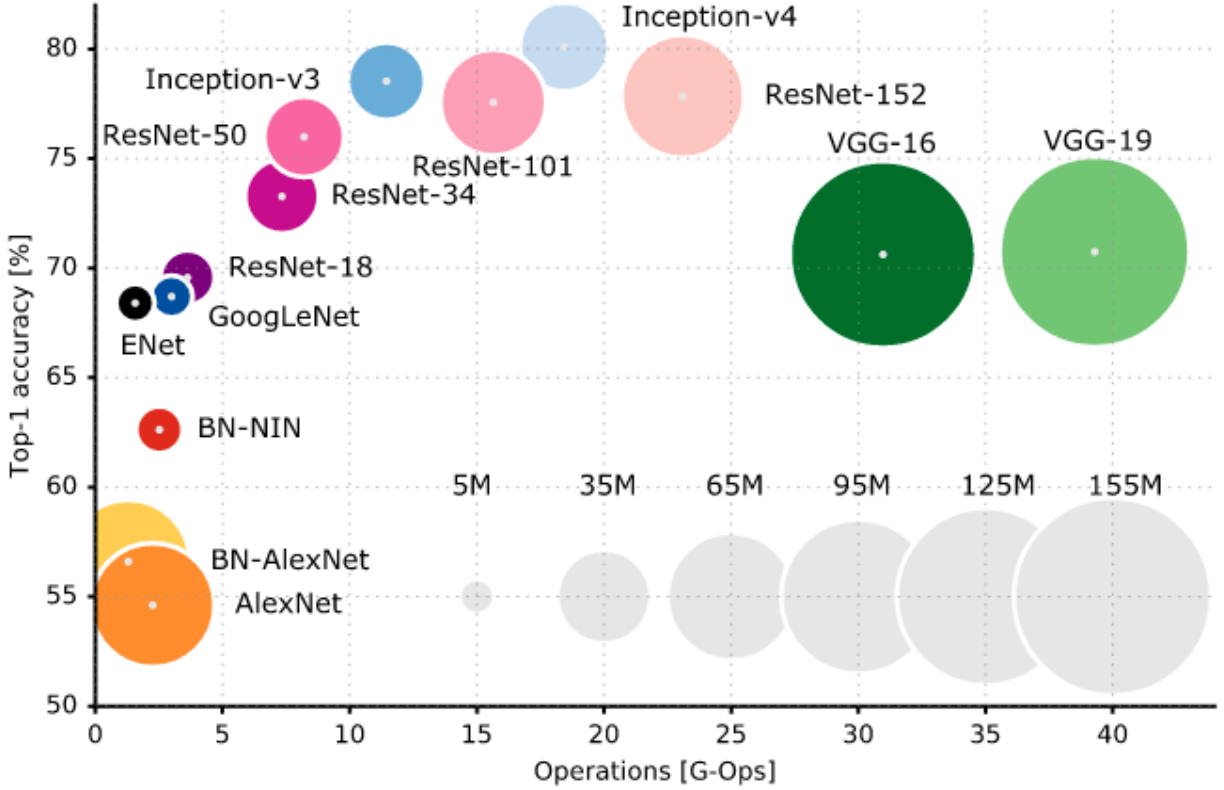


Taken from Stanford's CS231n: http://cs231n.github.io/classification/

Image Features



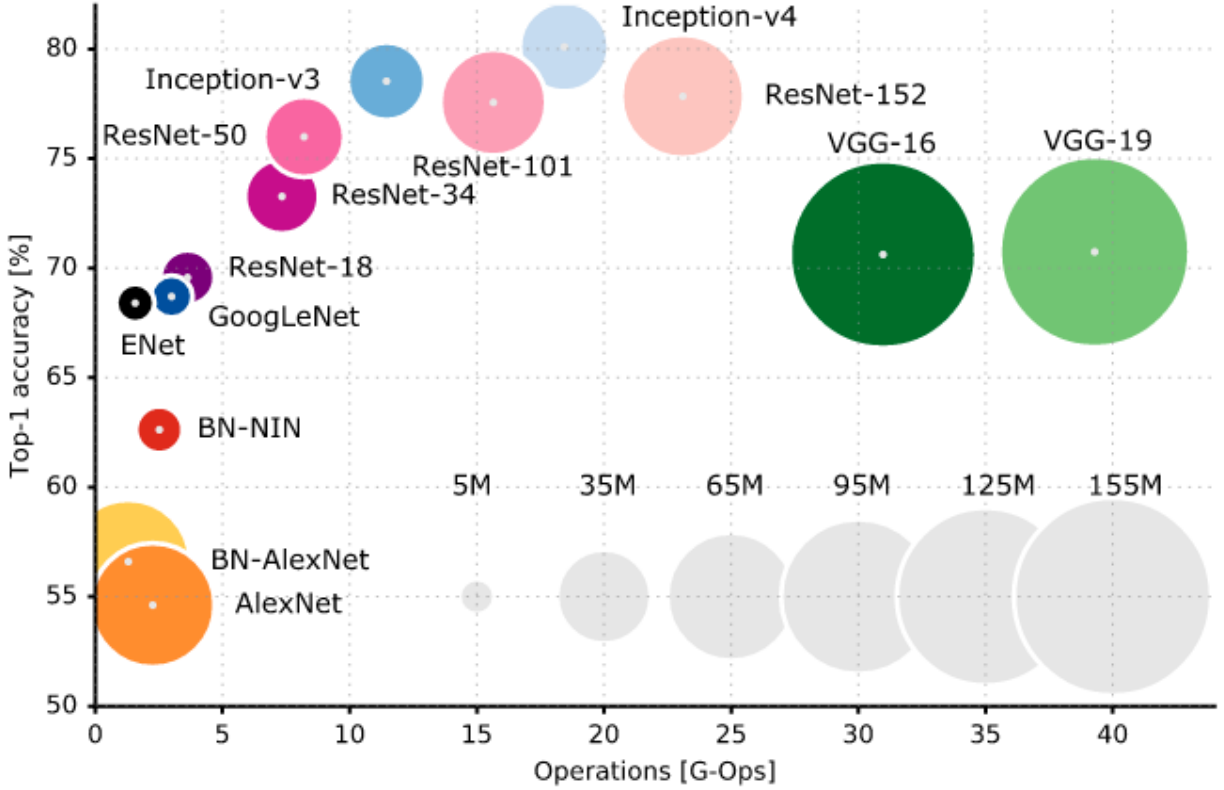
Choosing a Network



An Analysis of Deep Neural Network Models for Practical Applications

Alfredo Canziani, Eugenio Culurciello, Adam Paszke

Choosing a Network

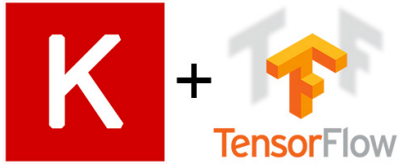


An Analysis of Deep Neural Network Models for Practical Applications

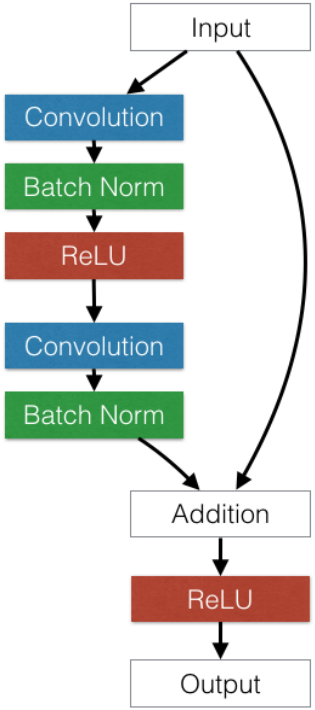
Alfredo Canziani, Eugenio Culurciello, Adam Paszke

ResNet50 seems like a good choice!

Keras Pre-Trained Models




```
from keras.applications import resnet50

model = resnet50.ResNet50(include_top=False,
                          weights='imagenet',
                          input_shape=(224, 224, 3))
```



x 50

ResNet50 Features

Extract the image files from our DataFrame.

```
# Read data and setup parameters
image_list = df['image_file'].tolist()
batch_size = 32
steps = int(np.ceil(len(image_list) / batch_size))

def imagenet_scale(x):
    """ImageNet is trained with the following mean pixels.
    """
    x[:, :,
    x[:, :,
    x[:, :,

    return
# Build a generator to feed data to the model
datagen = image_generators.ImageListDataGenerator(
    preprocessing_function=imagenet_scale)

generator = datagen.flow_from_image_list(
    image_list, y=
    image_dir=
    target_size=(
    batch_size=batch_size, shuffle=
# Predict
resnet_features =  model.predict_generator(
    generator,
    steps=steps)[:
```

ResNet50 Features

Apply the same pre-processing used to train the model

```
# Read data and setup parameters
image_list = df[
batch_size =
steps =

def imagenet_scale(x):
    """ImageNet is trained with the following mean pixels.
    """
    x[:, :, 0] -= 103.939
    x[:, :, 1] -= 116.779
    x[:, :, 2] -= 123.68

    return x

# Build a generator to feed data to the model
datagen = image_generators.ImageListDataGenerator(
    preprocessing_function=imagenet_scale)

generator = datagen.flow_from_image_list(
    image_list, y=
    image_dir=
    target_size=(
    batch_size=batch_size, shuffle=

# Predict
resnet_features =  model.predict_generator(
    generator,
    steps=steps)[:
```

ResNet50 Features

Loads the data. See accompanying repo if you want to use it.

```
# Read data and setup parameters
image_list = df[
batch_size =
steps =

def imagenet_scale(x):
    """ImageNet is trained with the following mean pixels.
    """
    x[:, :,
    x[:, :,
    x[:, :,

    return

# Build a generator to feed data to the model
datagen = image_generators.ImageListDataGenerator(
    preprocessing_function=imagenet_scale)

generator = datagen.flow_from_image_list(
    image_list, y=None,
    image_dir='images',
    target_size=(224, 224),
    batch_size=batch_size, shuffle=False)

# Predict
resnet_features =  model.predict_generator(
    generator,
    steps=steps)[:
```

ResNet50 Features

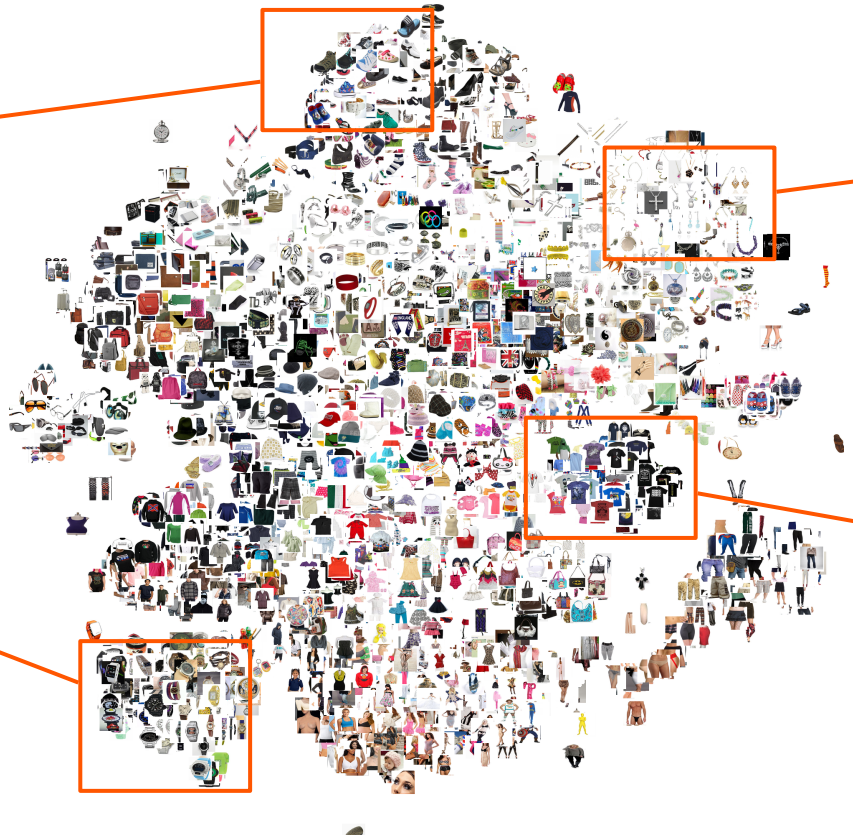
Make predictions

```
# Read data and setup parameters
image_list = df[
batch_size =
steps =

def imagenet_scale(x):
    """ImageNet is trained with the following mean pixels.
    """
    x[:, :,
    x[:, :,
    x[:, :,

    return
# Build a generator to feed data to the model
datagen = image_generators.ImageListDataGenerator(
    preprocessing_function=imagenet_scale)

generator = datagen.flow_from_image_list(
    image_list, y=
    image_dir=
    target_size=(
    batch_size=batch_size, shuffle=
# Predict
resnet_features =  model.predict_generator(
    generator,
    steps=steps)[:len(image_list)]
```

Image Embedding (SVD + t-SNE)

Shoes

Watches

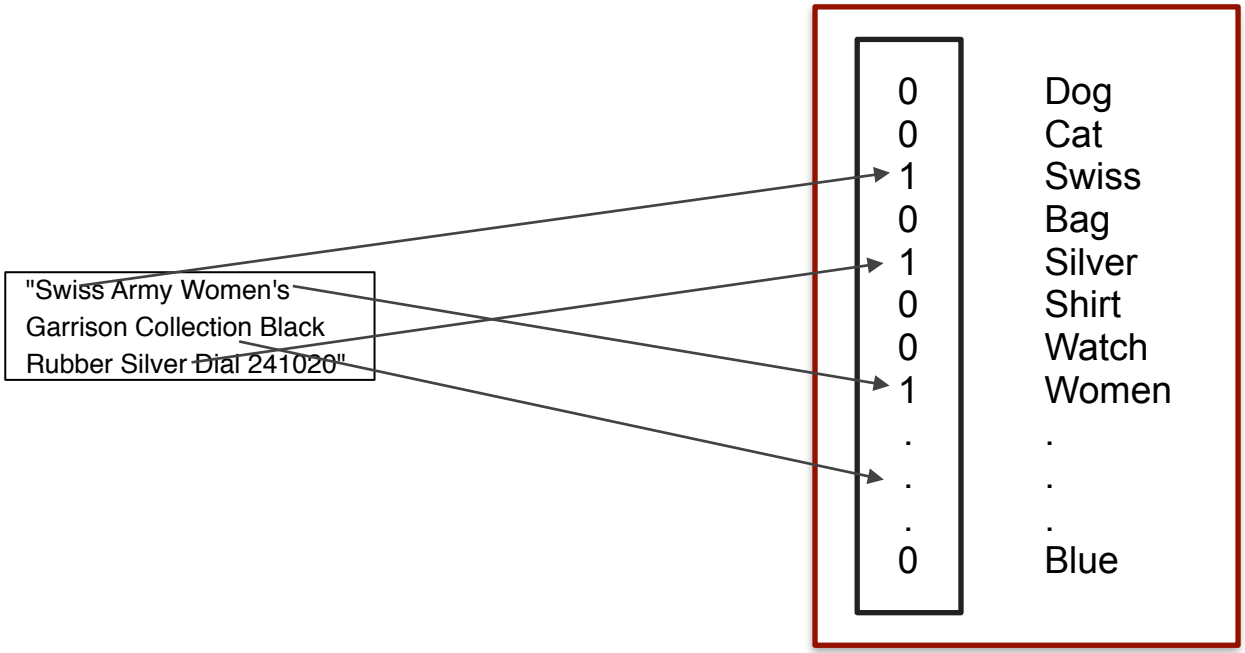


Jewelry

T-Shirts

Text

How is Text Represented?

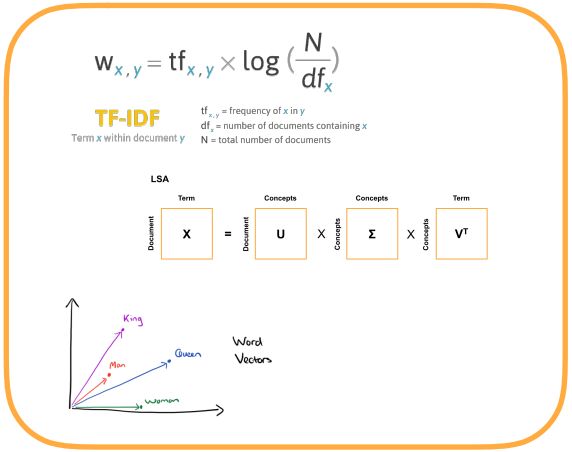


Text Features

Feature Extraction

Input

"Swiss Army Women's Garrison Collection Black Rubber Silver Dial 241020"



MODEL

Clothing

Accessory

Bag-of-Words in Python

"Swiss Army Women's Garrison Collection Black Rubber Silver Dial 241020"

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents

Bag-of-Words in Python

"Swiss Army Women's Garrison Collection Black Rubber Silver Dial 241020"

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

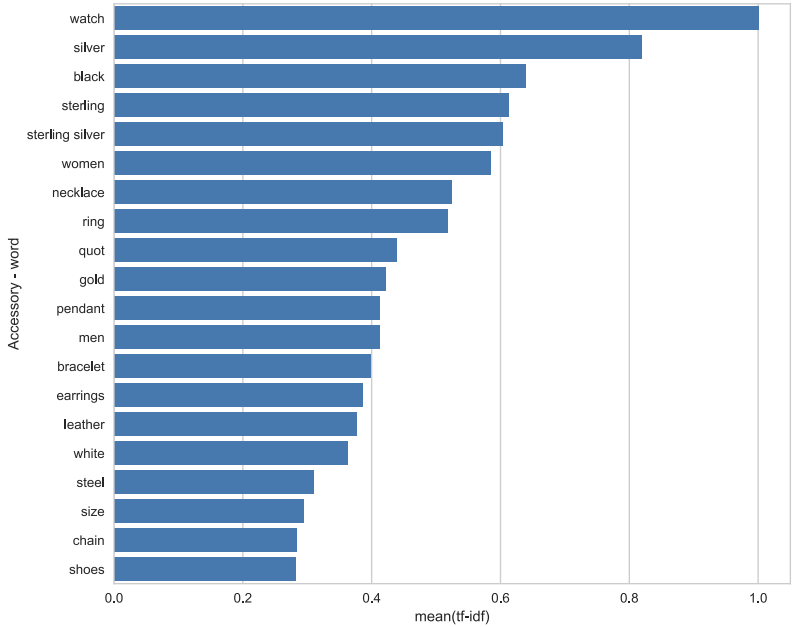
$tf_{x,y}$ = frequency of x in y

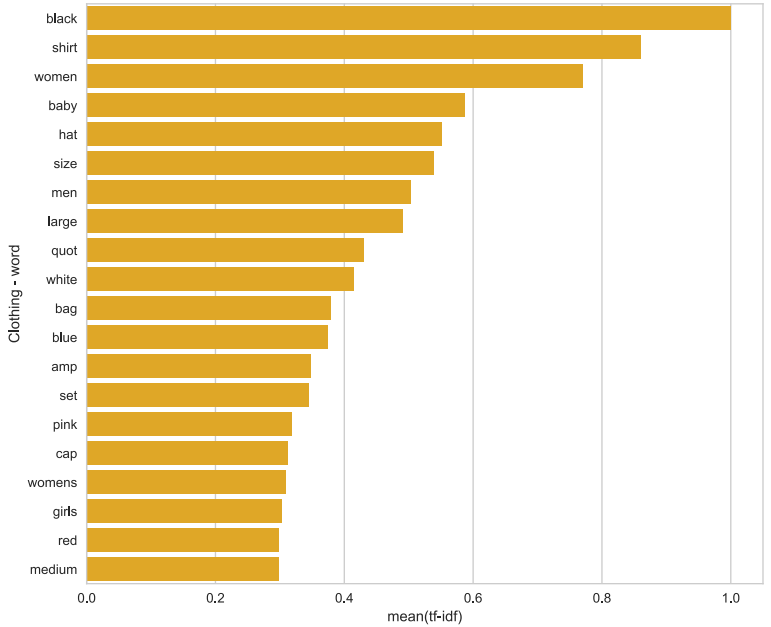
df_x = number of documents containing x

N = total number of documents

```
from sklearn.feature_extraction.text import (
    TfidfVectorizer)


text_list = df['title'].tolist()

vectorizer = TfidfVectorizer(ngram_range=(1,2),
                             stop_words='english',
                             sublinear_tf=False,
                             use_idf=True,
                             max_df=0.95,
                             min_df=5,
                             max_features=max_words)
tfidf = vectorizer.fit_transform(text_list)
```

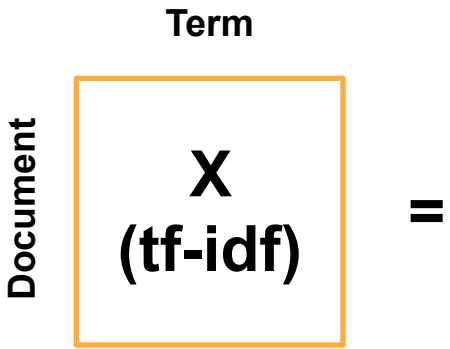
Bag-of-Words in Python





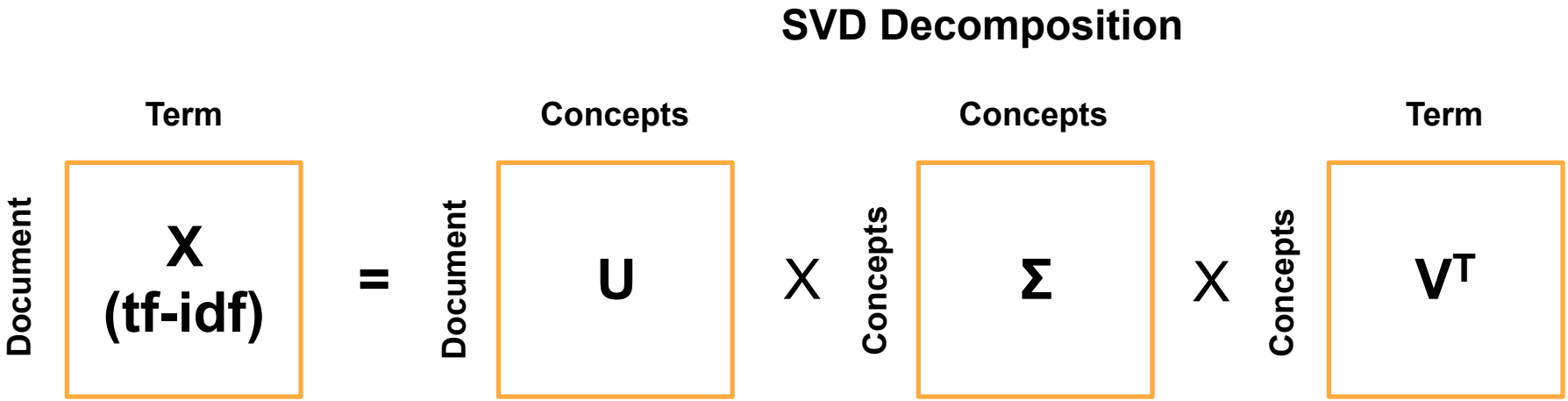
Word Vectors with LSA

Latent Semantic Analysis: Let's decompose our tf-idf features in a way that exposes the semantic meaning of the items. In this way we can obtain a more global understanding of our dataset.



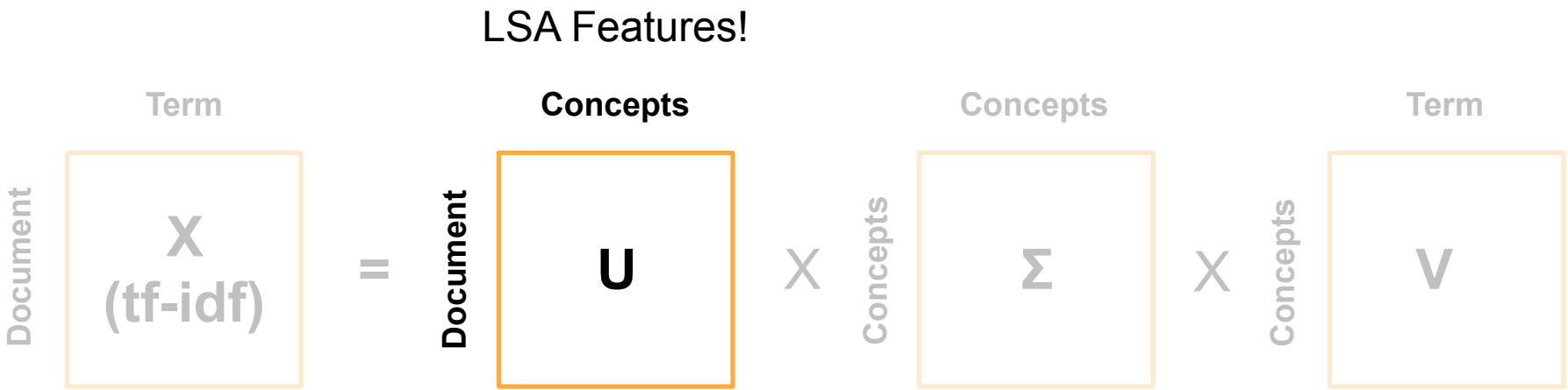
Word Vectors with LSA

Latent Semantic Analysis: Let's decompose our tf-idf features in a way that exposes the semantic meaning of the items. In this way we can obtain a more global understanding of our dataset.



Word Vectors with LSA

Latent Semantic Analysis: Let's decompose our tf-idf features in a way that exposes the semantic meaning of the items. In this way we can obtain a more global understanding of our dataset.



Word Vectors with LSA

TF-IDF transform text with sub-linear scaling.

```
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import Normalizer
from sklearn.pipeline import make_pipeline

# vectorizer text
vectorizer = TfidfVectorizer(ngram_range=(1,2),
                             stop_words='english',
                             sublinear_tf=True,
                             use_idf=True,
                             norm='l2'
                             max_features=10000)


# LSA Pipeline
svd = TruncatedSVD(n_components=
lsa = make_pipeline(vectorizer, svd)


# fit lsa
X = lsa.fit_transform(text_list)
```

Word Vectors with LSA

Use a scikit-learn pipeline to combine the TF-IDF and SVD steps.

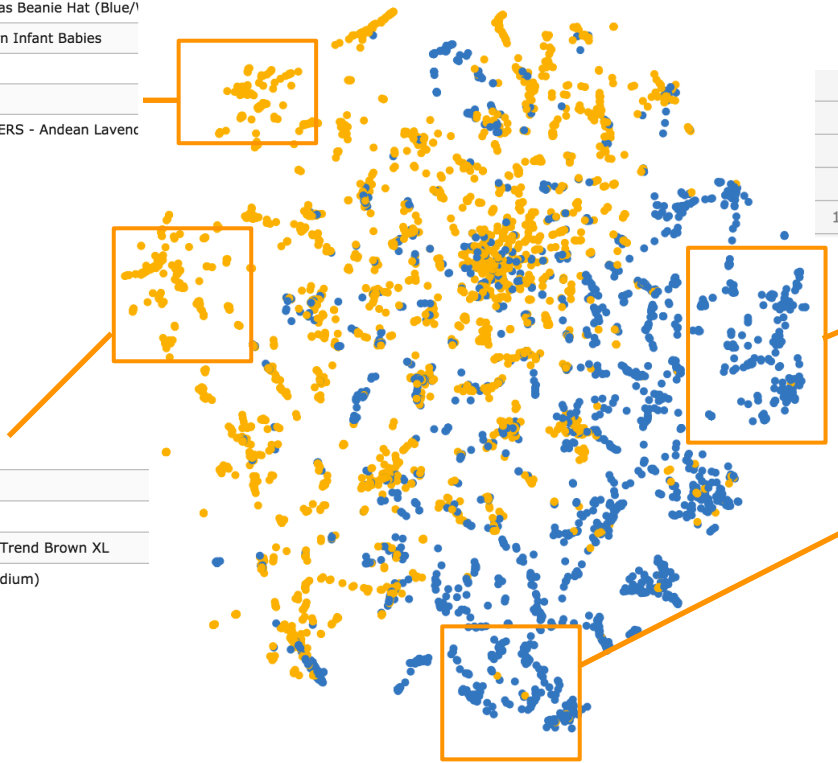
```
from sklearn.decomposition
from sklearn.preprocessing
from sklearn.pipeline

# vectorizer text
vectorizer = TfidfVectorizer(ngram_range=(
                                stop_words=
                                sublinear_tf=
                                use_idf=
                                norm=
                                max_features=


# LSA Pipeline
svd = TruncatedSVD(n_components=50, random_state=123)
lsa = make_pipeline(vectorizer, svd)


# fit lsa
X = lsa.fit_transform(text_list)
```

Word Vectors with LSA

We can use it like any other scikit-learn transformer!

```
from sklearn.decomposition
from sklearn.preprocessing
from sklearn.pipeline

# vectorizer text
vectorizer = TfidfVectorizer(ngram_range=(
                                stop_words=
                                sublinear_tf=
                                use_idf=
                                norm=
                                max_features=


# LSA Pipeline
svd = TruncatedSVD(n_components=
lsa = make_pipeline(vectorizer, svd)


# fit lsa
X = lsa.fit_transform(text_list)
```

Word Vectors with LSA (SVD + t-SNE)



15	Hotportgift Newborn Baby Girl Boy Cute Crochet Knit X-Mas Beanie Hat (Blue/
27	~ADORABLE~ Pink Crochet Beanie Flower Hat for Newborn Infant Babies
30	Kangol Patched Flexfit Baseball Hat,Tan,L/XL US
80	Magic Washed Military Hat
249	Handmade Lightweight Alpaca Hat with Earflaps for TODLERS - Andean Lavenc

Hats

52	Sterling Silver Stripes & Oval Ring Polished finish 3/4 inch wide, sizes 6 -
69	Classy Sterling Silver New Antiqued Cigarette Holder
82	925 St Sterling Silver Man On The Crescent Moon Pendant Necklace New Jewe
88	Crystal Clip On Earrings-White Gold Filled Crystal Pave Round Shape 20mm Cli
127	New 5mm Titanium Tension Set Ring, Pink Sapphire Bands

Jewelry

36	Ladies Sava(TM) Silk Shirt (up to size 2X)
73	YUUUP! T-shirt
103	Lito Boys Suit with Shirt, Vest & Tie (5 Piece)
113	Uoften Women Chiffon Tiered Sundress top Shirt Blouse Trend Brown XL
280	Pink Floyd - Dark Side of the Moon - Junior's T-Shirt (Medium)

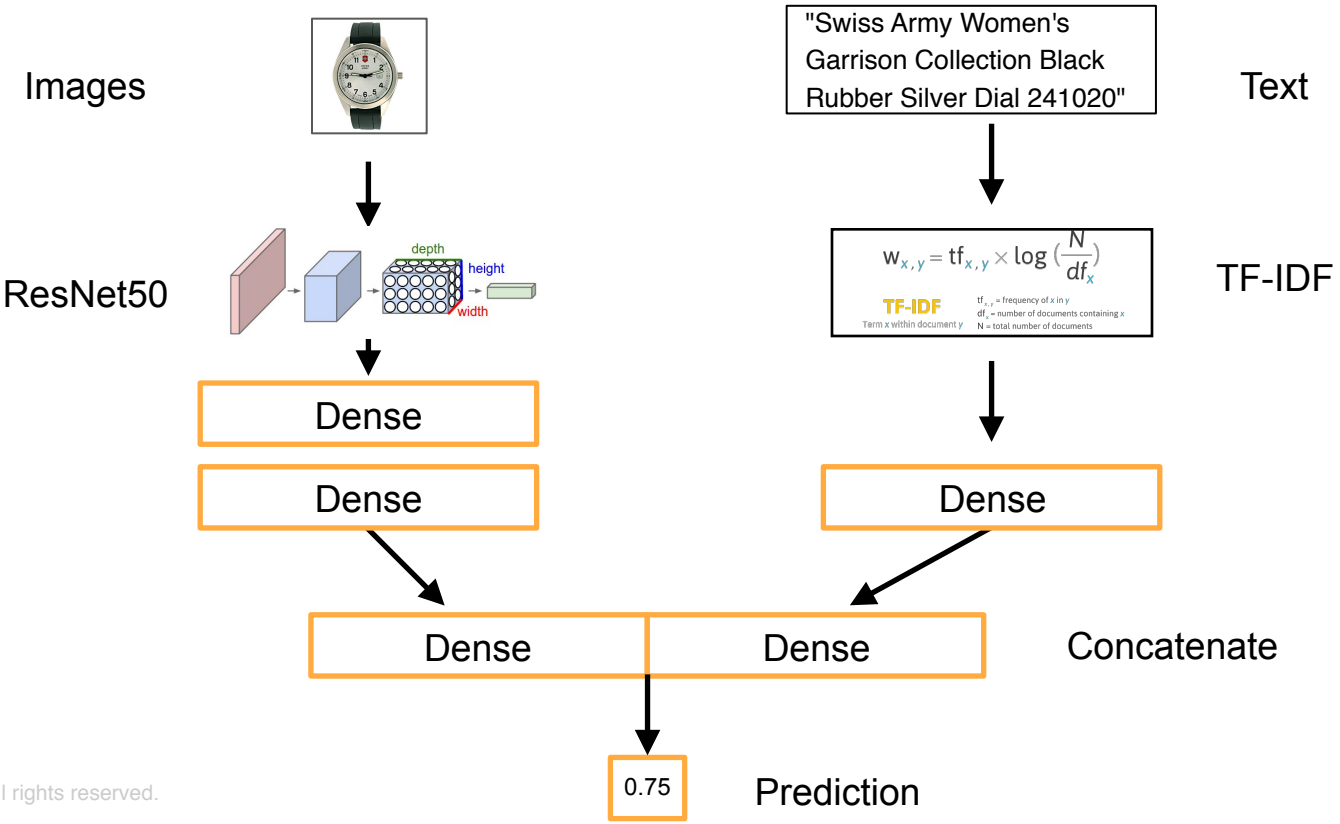
T-Shirts

20	Coil Watch Md Assorted
30	Ouku Christmas Gift Black Water Resistant Unisex Blue LED Digital Plastic Ban
40	GP Designs Women's Rhinestone-accented Link Watch
86	Ouku Low Price High Quality Women's Czechic Diamond Decoration Black Blue
97	Luxury Hip Hop Watch - Heavy Bling - Gunmetal Iced Out - with Leather Strap

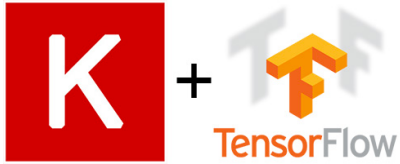
Watches

Modeling

Neural Network Model



Keras Workflow



Keras Implementation

We will use the Keras 2.0 Functional API

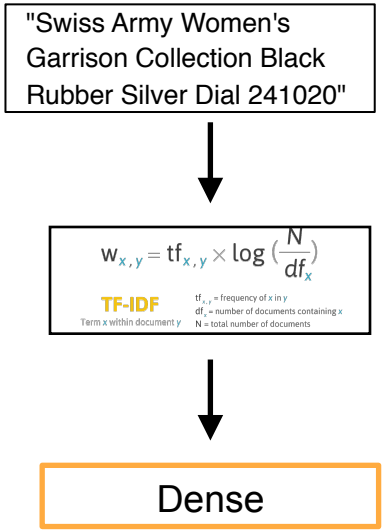
```
from keras.models import Model
from keras.layers import Input, Dense
from keras.layers import Dropout, concatenate

# input variables
text_features = Input(shape=tfidf.shape[
                      dtype=
image_features = Input(shape=resnet_features.shape[
                       dtype=

# text model
x_text = Dense(
x_text = Dropout(

# image model
x_img = Dense(
x_img = Dropout(
x_img = Dense(
x_img = Dropout(

# concatenate image and text vectors
merged
predictions = Dense(

# choose optimizer and compile
model = Model(inputs=[image_features, text_features],
              outputs=[predictions])
model.compile(optimizer=
              loss=
              metrics=[
```

Keras Implementation

TensorFlow uses a static computational graph, so we need to specify the inputs up-front.

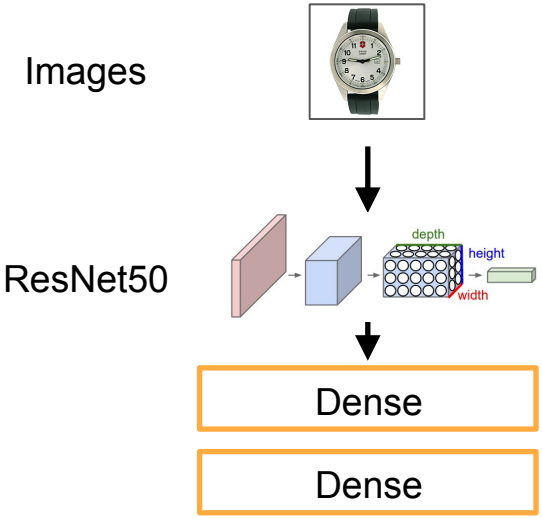
```
from keras.models
from keras.layers
from keras.layers

# input variables
text_features = Input(shape=tfidf.shape[1:],
                      dtype='float32')
image_features = Input(shape=resnet_features.shape[1:],
                       dtype='float32')

# text model
x_text = Dense(
x_text = Dropout(

# image model
x_img = Dense(
x_img = Dropout(
x_img = Dense(
x_img = Dropout(

# concatenate image and text vectors
merged
predictions = Dense(

# choose optimizer and compile
model = Model(inputs=[image_features, text_features],
              outputs=[predictions])
model.compile(optimizer=
              loss=
              metrics=[
```

Keras Implementation



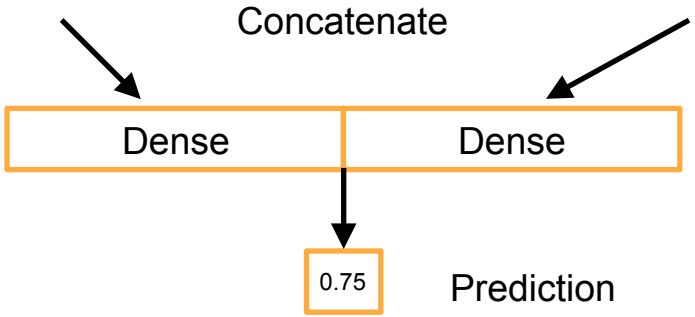

```
from keras.models
from keras.layers
from keras.layers

# input variables
text_features = Input(shape=tfidf.shape[
                      dtype=
image_features = Input(shape=resnet_features.shape[
                       dtype=

# text model
x_text = Dense(256, activation='relu')(text_features)
x_text = Dropout(0.5)(x_text)

# image model
x_img = Dense(
x_img = Dropout(
x_img = Dense(
x_img = Dropout(

# concatenate image and text vectors
merged
predictions = Dense(

# choose optimizer and compile
model = Model(inputs=[image_features, text_features],
              outputs=[predictions])
model.compile(optimizer=
              loss=
              metrics=[
```

Keras Implementation




```
from keras.models
from keras.layers
from keras.layers

# input variables
text_features = Input(shape=tfidf.shape[
                      dtype=
image_features = Input(shape=resnet_features.shape[
                       dtype=


# text model
x_text = Dense(
x_text = Dropout(

# image model
x_img = Dense(256, activation='relu')(image_features)
x_img = Dropout(0.5)(x_img)
x_img = Dense(256, activation='relu')(x_img)
x_img = Dropout(0.5)(x_img)

# concatenate image and text vectors
merged
predictions = Dense(

# choose optimizer and compile
model = Model(inputs=[image_features, text_features],
              outputs=[predictions])
model.compile(optimizer=
              loss=
              metrics=[
```

Keras Implementation




```
from keras.models
from keras.layers
from keras.layers

# input variables
text_features = Input(shape=tfidf.shape[
                      dtype=
image_features = Input(shape=resnet_features.shape[
                       dtype=


# text model
x_text = Dense(
x_text = Dropout(

# image model
x_img = Dense(
x_img = Dropout(
x_img = Dense(
x_img = Dropout(

# concatenate image and text vectors
merged = concatenate([x_img, x_text])
predictions = Dense(1, activation='sigmoid')(merged)

# choose optimizer and compile
model = Model(inputs=[image_features, text_features],
              outputs=[predictions])
model.compile(optimizer=
              loss=
              metrics=[
```

Keras Implementation

Since we have two classes, we use log-loss as our objective. A good default optimizer is Adam.

```
from keras.models
from keras.layers
from keras.layers

# input variables
text_features = Input(shape=tfidf.shape[
                      dtype=
image_features = Input(shape=resnet_features.shape[
                       dtype=

# text model
x_text = Dense(
x_text = Dropout(

# image model
x_img = Dense(
x_img = Dropout(
x_img = Dense(
x_img = Dropout(

# concatenate image and text vectors
merged
predictions = Dense(

# choose optimizer and compile
model = Model(inputs=[image_features, text_features],
              outputs=[predictions])
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Fitting the Model

Keras expects a binary matrix of labels

Accessory → [1, 0]
Clothing → [0, 1]

```
from sklearn.preprocessing import LabelBinarizer

# encode labels (binary labels)
encoder = LabelBinarizer()
train_labels = encoder.fit_transform(
   train_df.pop('product_categories'))
test_labels = encoder.transform(
   test_df.pop('product_categories'))


# setup a generator for sparse data
generator = sparse_batch_generator(
    resnet_features,
    tfidf,
    train_labels,
    shuffle=

# fit the model
batch_size =
steps_per_epoch =
model.fit_generator(
    generator,
    steps_per_epoch=steps_per_epoch,
    epochs=
    validation_data=[
      [test_resnet_features, test_tfidf], test_labels])

model.save(
```

Fitting the Model

This generator is responsible for batching out the features. See accompanying repo for details.

```
from sklearn.preprocessing

# encode labels (binary labels)
encoder = LabelBinarizer()
train_labels = encoder.fit_transform(
   train_df.pop(
test_labels = encoder.transform(
   test_df.pop(


# setup a generator for sparse data
generator = sparse_batch_generator(
    resnet_features,
    tfidf,
    train_labels,
    shuffle=True)

# fit the model
batch_size =
steps_per_epoch =
model.fit_generator(
    generator,
    steps_per_epoch=steps_per_epoch,
    epochs=
    validation_data=[
      [test_resnet_features, test_tfidf], test_labels])

model.save(
```

Fitting the Model

Call fit

```
Epoch 1/50
324/324 [==============================] - 6s - loss: 0.3252 - acc: 0.8678 - val_loss: 0.2280 - val_acc: 0.9035
Epoch 2/50
324/324 [==============================] - 2s - loss: 0.1689 - acc: 0.9359 - val_loss: 0.2043 - val_acc: 0.9213
Epoch 3/50
324/324 [==============================] - 2s - loss: 0.1241 - acc: 0.9565 - val_loss: 0.2195 - val_acc: 0.9205
Epoch 4/50
324/324 [==============================] - 2s - loss: 0.1014 - acc: 0.9676 - val_loss: 0.2462 - val_acc: 0.9144
Epoch 5/50
324/324 [==============================] - 2s - loss: 0.0844 - acc: 0.9760 - val_loss: 0.2571 - val_acc: 0.9155
Epoch 6/50
324/324 [==============================] - 2s - loss: 0.0747 - acc: 0.9802 - val_loss: 0.2752 - val_acc: 0.9147
Epoch 7/50
324/324 [==============================] - 2s - loss: 0.0648 - acc: 0.9838 - val_loss: 0.2979 - val_acc: 0.9113
Epoch 8/50
324/324 [==============================] - 2s - loss: 0.0552 - acc: 0.9874 - val_loss: 0.3266 - val_acc: 0.9128
Epoch 9/50
324/324 [==============================] - 2s - loss: 0.0521 - acc: 0.9877 - val_loss: 0.3418 - val_acc: 0.9074
Epoch 10/50
324/324 [==============================] - 2s - loss: 0.0488 - acc: 0.9898 - val_loss: 0.3668 - val_acc: 0.9055
```

But always remember to save your model!

```
from sklearn.preprocessing

# encode labels (binary labels)
encoder = LabelBinarizer()
train_labels = encoder.fit_transform(
   train_df.pop(
test_labels = encoder.transform(
   test_df.pop(


# setup a generator for sparse data
generator = sparse_batch_generator(
    resnet_features,
    tfidf,
    train_labels,
    shuffle=

# fit the model
batch_size = 32
steps_per_epoch = int(np.ceil(tfidf.shape[0]/batch_size))
model.fit_generator(
    generator,
    steps_per_epoch=steps_per_epoch,
    epochs=50,
    validation_data=[
      [test_resnet_features, test_tfidf], test_labels])

model.save('image_text_model.hdf5')
```

Model Evaluation

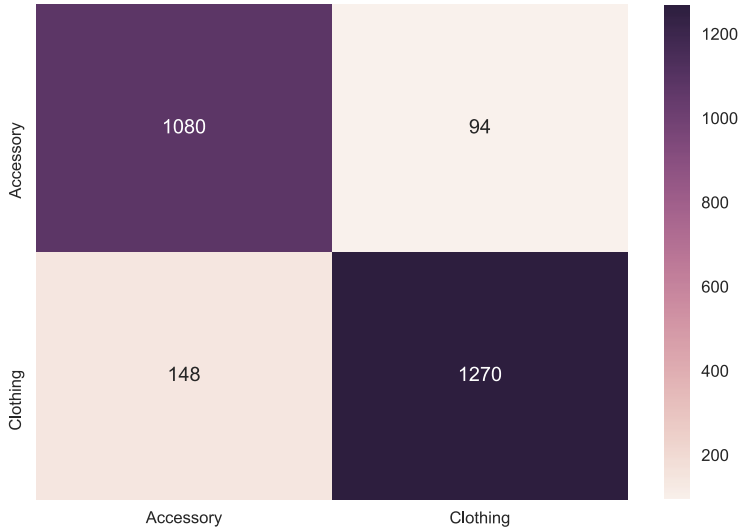
Was Our Model Any Good?

Compare Metrics with a Baseline

	accuracy	auc	log_loss	precision (true positive accuracy)	recall (positive accuracy)
name					
Majority Classifier	0.547068	0.500000	15.644083	0.547068	1.000000
Keras Model	0.906636	0.966134	0.314568	0.931085	0.895628

Was Our Model Any Good?

Confusion Matrix



How Can We Improve?

Leather Hat Key Ring (Model ~ Clothing)

How Can We Improve?

3 Rows Pyramid Stud Punk Spike Gothic Leather Belt Black, Small

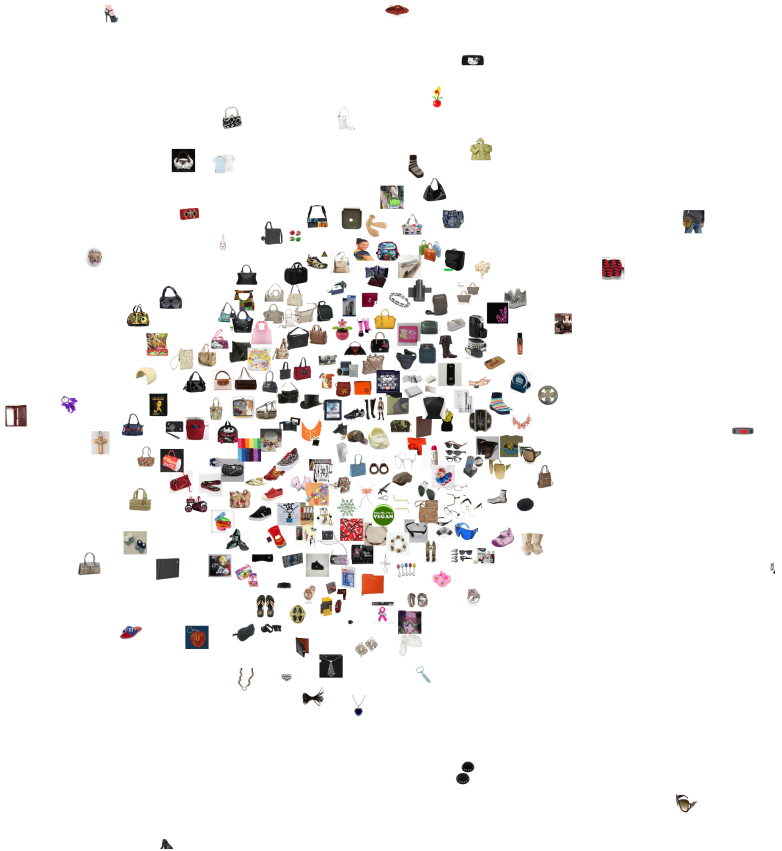
(Model ~ Accessory)

How Can We Improve?

Otium 21110CF Canvas Genuine Leather Cross Body Waist Bag Waist Pack Hipshot,Coffee’

(Model ~ Clothing)

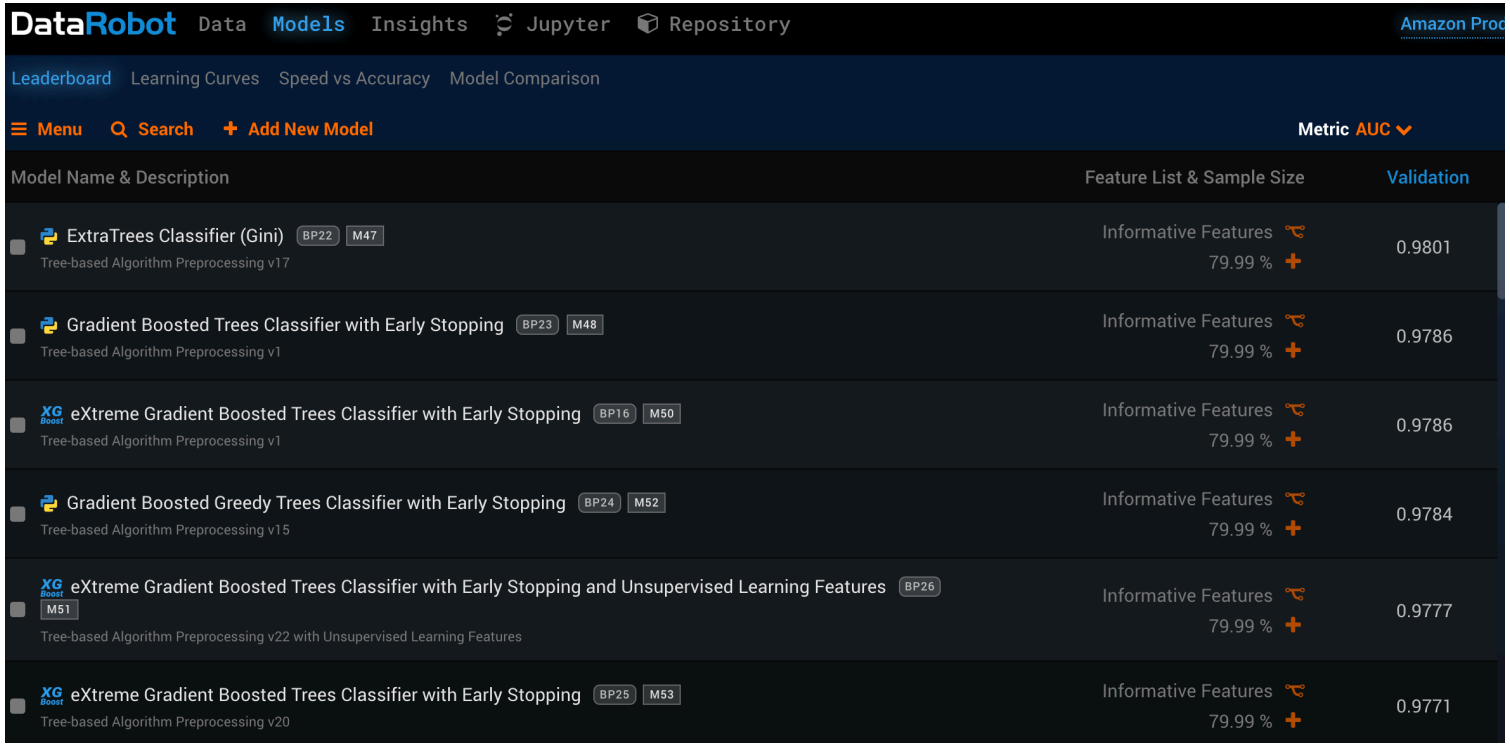
How Can We Improve?



Where to Next?

- Try different hyper-parameters
- Try fine-tuning other layers of ResNet50
- Try adding data augmentation
- Try different models
- ….

DataRobot Can Help!

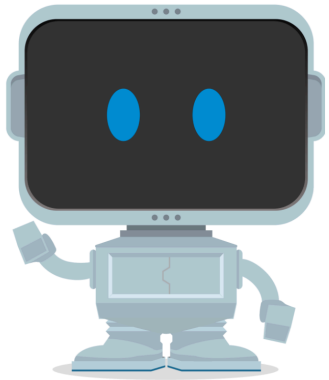


Questions?

Joshua Loyal: Data Scientist at DataRobot
- joshua@datarobot.com

Igor Veksler: Data Scientist at DataRobot
- igor.veksler@datarobot.com

Code for this talk can be found here
- https://github.com/joshloyal/pydata-amazon-products



Extras

Feeding Data Into the Network

```
def sparse_batch_generator(image_features, text_features, labels, batch_size=32, shuffle=False, random_state=123):
    random_state = np.random.RandomState(random_state)

    n_samples = image_features.shape[0]
    n_batches = np.ceil(n_samples / batch_size)

    counter = 0
    sample_index = np.arange(n_samples)
    if shuffle:
        random_state.shuffle(sample_index)

    while True:
        batch_index = sample_index[batch_size * counter:batch_size * (counter + 1)]

        image_batch = image_features[batch_index, :]
        if sparse.issparse(image_batch):
            image_batch = image_batch.toarray()

        text_batch = text_features[batch_index, :]
        if sparse.issparse(text_batch):
            text_batch = text_batch.toarray()

        labels_batch = labels[batch_index]
        counter += 1
        yield [image_batch, text_batch], labels_batch

        if counter == n_batches:
            if shuffle:
                random_state.shuffle(sample_index)
            counter = 0
```

Transfer Learning