**Breast Cancer Diagnosis and Classification**

**Background:**

Breast cancer is a leading cause of cancer death in women. 1 in 8 women are diagnosed with breast cancer at some point in their life. Early diagnosis and treatment are crucial to reduce the mortality rate and increase patients’ lifespan. Mammography is effective in early detection, reducing the mortality rate by 3-13%. Therefore, women above a certain age are recommended to get yearly mammography exams. With so many exams, radiologists must manually inspect each image to determine whether or not that woman has breast cancer. We can use machine learning to automate or speed up this process.

**Goal:**

Design a machine learning model to classify types of cancerous breast tissue from mammography images.

**Dataset:**

<https://goo.gl/GgLMrK>

**Plan:**

1. Load Data-turn into matrix
2. Extract features
3. Make model-Classification model
4. Train model
5. Output should be 1(true) or 0(false)
6. Test model

Extracting features:

Downloading trained CNN models: (run in a Python shell: type python into terminal)

from torchvision import models

vgg11\_model = models.vgg11(pretrained=True)

vgg13\_model = models.vgg13(pretrained=True)

resnet18\_model = models.resnet18(pretrained=True)

resnet34\_model = models.resnet34(pretrained=True)

Loading image into VGG11 model and extracting features:

import torch

import torch.nn as nn

import torchvision.transforms as transforms

from torchvision import models

from torch.autograd import Variable

imgs = np.array(imgs, dtype=float)

imgs = np.repeat(imgs, 3, axis=3)

imgs = imgs.transpose(0, 3, 1, 2)

imgs\_tensor = torch.Tensor(imgs)

normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],

std=[0.229, 0.224, 0.225])

imgs\_tensor = normalize(imgs\_tensor)

model = models.vgg11(pretrained=True)

# remove last fully-connected layer

new\_classifier = nn.Sequential(\*list(model.classifier.children())[:-1])

model.classifier = new\_classifier

outputs = model(Variable(imgs\_tensor))

outputs\_np = outputs.data.numpy() # Features!

Feature extraction:

* PCA - with different numbers of components
* Raw image

Data augmentation:

from sklearn import decomposition

X = imgs.reshape((322, -1))

pca = [decomposition.PCA](http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html#sklearn.decomposition.PCA)(n\_components=30)  
pca.fit(X)  
X = pca.transform(X) # features!!

Classification Models:

1. Logistic Regression: <http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html>

split\_ind = len(X) / 5

trainX = X[:4\*split\_ind]

trainy = labels[:4\*split\_ind]

testX = X[4\*split\_ind:]

testy = labels[4\*split\_ind:]

logreg = LogisticRegression()

logreg.fit(trainX,trainy)

score = logreg.score(testX,testy)

print(score)

1. Support Vector Machines: <http://scikit-learn.org/stable/modules/svm.html#classification>

supvec= svm.SVC()

supvec.fit(trainX,trainy)

score\_2 = supvec.score(testX,testy)

print(score\_2)

1. Nearest Neighbors: <http://scikit-learn.org/stable/modules/neighbors.html#nearest-neighbors-classification>

nbrs = neighbors.KNeighborsClassifier()

nbrs.fit(trainX,trainy)

score\_3 = nbrs.score(testX,testy)

print(score\_3)

1. Gaussian Process: <http://scikit-learn.org/stable/modules/gaussian_process.html#gaussian-process-classification-gpc>

gaussian = GaussianProcessClassifier()

gaussian.fit(trainX,trainy)

score\_4 = gaussian.score(testX,testy)return (score\_4)

print(score\_4)

1. Decision Tree: <http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier>

dctree= DecisionTreeClassifier()

dctree.fit(trainX,trainy)

score\_5 = dctree.score(testX,testy)

print(score\_5)

**Results:**

Evaluation: precision, recall, ROC curves, cross validation

Validation curve: <http://scikit-learn.org/stable/auto_examples/model_selection/plot_validation_curve.html#sphx-glr-auto-examples-model-selection-plot-validation-curve-py>

Train vs. Test error: <http://scikit-learn.org/stable/auto_examples/model_selection/plot_train_error_vs_test_error.html#sphx-glr-auto-examples-model-selection-plot-train-error-vs-test-error-py>

ROC curve: (classification only) <http://scikit-learn.org/stable/auto_examples/model_selection/plot_roc_crossval.html#sphx-glr-auto-examples-model-selection-plot-roc-crossval-py>

Cross validation:

<http://scikit-learn.org/stable/modules/cross_validation.html>

**def feature\_extract\_cnn(imgs, model\_fn, batch\_size=50):**

**imgs = np.array(imgs, dtype=float)**

**imgs = np.repeat(imgs, 3, axis=3)**

**imgs = imgs.transpose(0, 3, 1, 2)**

**all\_outputs = []**

**num\_batches = int(math.ceil(len(imgs) / batch\_size))**

**model = model\_fn(pretrained=True)**

**# remove last fully-connected layer**

**try:**

**new\_classifier = nn.Sequential(\*list(model.classifier.children())[:-1])**

**model.classifier = new\_classifier**

**except:**

**new\_classifier = nn.Sequential(\*list(model.fc.children())[:-1])**

**model.fc = new\_classifier**

**for nb in range(num\_batches):**

**print(nb)**

**start = nb \* batch\_size**

**end = (nb + 1) \* batch\_size**

**img\_batch = imgs[start:end]**

**imgs\_tensor = torch.Tensor(img\_batch)**

**normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],**

**std=[0.229, 0.224, 0.225])**

**imgs\_tensor = normalize(imgs\_tensor)**

**outputs = model(Variable(imgs\_tensor))**

**outputs\_np = outputs.data.numpy() # Features!**

**all\_outputs.append(outputs\_np)**

**return np.concatenate(all\_outputs)**