Capstone Project: ADNI Data - Markdown 2

Brian Collica, Ben Searchinger, Ryan Roggenkemper, James Koo

3/30/2021

Model Description

Our model consists of two stages. Stage one tries to predict β-amyloid positivity for a given patient. Research has shown that a β-amyloid positive status is highly correlated with Alzheimer's Disease, but testing for positivity is costly and invasive to the patient. We hope to cut down on these costs by accurately predicting a patient's status using other easily accessible data.

Stage two is a deep learning classifier that is fit to tau PET brain scans in order to classify a patient as having AD-related tau pathology or normal tau levels. The spatial arrangement of the tau protein in the brain has been a topic of recent AD research, and we hope that the deep learning classifier can pick up on these subtle arrangements.

The ultimate goal is to effectively sort patients which are at high risk for AD using the first stage, and then accurately classify their AD status based on the PET brain scans. The idea is that a patient would only receive a PET scan if they are labeled as β-amyloid in the first stage. If successful, we hope this technique can cut down on unnecessary costly procedures which are invasive and potentially harmful to the patient while also maintaining a high level of accuracy in quantifying an individual's AD risk.

We plan to evaluate our success with the following metrics:

- First Stage True Positive, False Negative, Misclassification Rates, and Correlation with AD Diagnosis.
- Second Stage Accuracy, True Positive, False Negative, & Misclassification Rates
- Combined Model Accuracy, True Positive, False Negative, & Misclassification Rates

Ideally, we want a high proportion of the subjects predicted as β-amyloid positive to be diagnosed with AD or MCI. In particular, we want the proportion of AD/MCI subjects among those predicted as β-amyloid positive to be higher than the proportion in the general population. That being said, some other metrics of interest will be the distribution of characteristics among those labeled as β-amyloid negative in the first stage, and if there is any particular information which can be used to fine tune the model.

First Stage Initial Fits

Description

For the first stage model, we used the ADNIMERGE dataset available on the ADNI website. This dataset includes many key variables of interest for each ADNI participant. The dataset was filtered to include only patients who had cerebrospinal-fluid measurements taken at baseline. Each patient's amyloid measurements were measured in a lab at the University of Pennsylvania and included in the UPENNBIOMK_MASTER dataset, also available on the ADNI website. We combined the two datasets to get 1,087 unique baseline observations.

The two initial models considered were (1) a linear regression to predict `beta_csf` - the specific level of β-amyloid for a given patient, and (2) a logistic regression to predict positivity status, `beta_pos`. The variable `beta_pos` was coded as 1 if an individual had `beta_csf` less than 192 and 0 otherwise. The level of 192 is a well-established clinical cutoff point for determining β-amyloid positivity. In the linear regression model, patients were classified as β-amyloid positive if their *predicted* amyloid level was below 192.

The linear and logistic models were both fit using the following variables known to be linked to AD:

- Age
- Sex (1 for Male 0 for Female)
- Immediate Memory Recall Score in the Rey Auditory Verbal Learning Test (RAVLT)
- Presence of APO$\epsilon - 4$ Gene
  - `APOE$_1` $= 1$ if only one allele is present
  - `APOE$_2` $= 1$ if two alleles are present

Both models were also fit again with the addition of two variables containing brain volumetrics estimated from MRI scans:

- Whole Brain Volume
- Hippocampus Volume

Code and Output

The first stage models are fit using R and requires the following packages: `readr`, `dplyr`, `stringr`, and `origami`. The following code reads in the data, filters by baseline visit, and renames the baseline variables of interest.

```
# Load Packages ####
library(readr)
library(dplyr)
library(stringr)
library(origami)

# Read Data ####
beta_tau <- read_csv("AmyloidData/beta_tau_csf_new_vars.csv")
source("cv_functions.R")

# Filter By Baseline Visit ####
baseline <- beta_tau %>%
  filter(VISCODE == "bl")

bl_na1 <- which(is.na(baseline$beta_pos) | is.na(baseline$RAVLT_immediate_bl) |
                  is.na(baseline$Hippocampus) | is.na(baseline$WholeBrain))
bl_na2 <- which(is.na(baseline$beta_pos) | is.na(baseline$RAVLT_immediate_bl))

# Rename _bl Variables ####
# Rename _bl Variables ####
bl_rm <- c(27, 54, 55)
baseline_1 <- baseline[-(bl_na1), -(bl_rm)]
```

```
baseline_1 <- rename(
  baseline_1,
  RAVLT_immediate = RAVLT_immediate_bl,
  Hippocampus = Hippocampus_bl,
  WholeBrain = WholeBrain_bl)

baseline_2 <- baseline[-(bl_na2), -(bl_rm)]
baseline_2 <- rename(
  baseline_2,
  RAVLT_immediate = RAVLT_immediate_bl,
  Hippocampus = Hippocampus_bl,
  WholeBrain = WholeBrain_bl)
```

The linear and logistic fits using all the baseline data can be seen below.

```
# Initial Model Fits ####
# Including MRI data
bl_cont <- lm(beta_csf ~ AGE + MALE + APOE4_1 + APOE4_2 + RAVLT_immediate
              + Hippocampus + WholeBrain,
              data = baseline_1)
bl_logit <- glm(beta_pos ~ AGE + MALE + APOE4_1 + APOE4_2 + RAVLT_immediate
                + Hippocampus + WholeBrain,
                data = baseline_2, family = "binomial")

# Excluding MRI Data
bl_cont2 <- lm(beta_csf ~ AGE + MALE + APOE4_1 + APOE4_2 + RAVLT_immediate,
               data = baseline_1)
bl_logit2 <- glm(beta_pos ~ AGE + MALE + APOE4_1 + APOE4_2 + RAVLT_immediate,
                 data = baseline_2, family = "binomial")
```

5-fold crass-validation was then run for each of the four initial baseline models. The following code performs the cross-validation and also calculates performance metrics of interest. The metrics calculated include:

- Misclassification Rate: Proportion of incorrect predictions
- True Positive Rate: Proportion of β-amyloid positive cases correctly classified
- False Positive Rate: Proportion of β-amyloid negative cases classified as positive
- True Negative Rate: Proportion of β-amyloid negative cases correctly classified
- False Negative Rate: Proportion of β-amyloid positive cases classified as negative
- DX Correlation: Correlation between positive prediction status and being diagnosed as AD at baseline
- Positive AD: Number of subjects predicted as β-amyloid positive who were also diagnosed with AD
- Negative AD: Number of subjects predicted as β-amyloid negative who were also diagnosed with AD
- Percent Positive AD: Percentage of positive predictions who were also diagnosed with AD

Specific functions needed for the cross-validation and metrics calculations are defined in the `cv_functions.R` script.

```r
# 5-Fold Cross Validation ####
# Make Folds
bl_folds1 <- folds_vfold(nrow(baseline_1), V = 5)
bl_folds2 <- folds_vfold(nrow(baseline_2), V = 5)

# Linear Models ####
bl_cv_lm <- origami::cross_validate(
  cv_fun = cv_lm, folds = bl_folds1, data = baseline_1,
  reg_form = "beta_csf ~ AGE + MALE + APOE4_1 + APOE4_2 + RAVLT_immediate + Hippocampus + WholeBrain")

bl_cv_lm2 <- origami::cross_validate(
  cv_fun = cv_lm, folds = bl_folds2, data = baseline_2,
  reg_form = "beta_csf ~ AGE + MALE + APOE4_1 + APOE4_2 + RAVLT_immediate")

# Logistic Models ####
bl_cv_logit <- origami::cross_validate(
  cv_fun = cv_logit, folds = bl_folds1, data = baseline_1,
  reg_form = "beta_pos ~ AGE + MALE + APOE4_1 + APOE4_2 + RAVLT_immediate + Hippocampus + WholeBrain")

bl_cv_logit2 <- origami::cross_validate(
  cv_fun = cv_logit, folds = bl_folds2, data = baseline_2,
  reg_form = "beta_pos ~ AGE + MALE + APOE4_1 + APOE4_2 + RAVLT_immediate")

# Linear Models Stats ####
bl_lm_stats1 <- colMeans(dplyr::bind_rows(bl_cv_lm$c_stats))
bl_lm_stats2 <- colMeans(dplyr::bind_rows(bl_cv_lm2$c_stats))

# Logistic Model Stats ####
bl_log_stats1 <- colMeans(dplyr::bind_rows(bl_cv_logit$c_stats))
bl_log_stats2 <- colMeans(dplyr::bind_rows(bl_cv_logit2$c_stats))

# Combine Data ####
bl_summary <- dplyr::bind_rows(
        bl_lm_stats1, bl_lm_stats2, bl_log_stats1, bl_log_stats2)
```

Table 1: Cross-Validation Results

	Misclass	T Pos	F Pos	T Neg	F Neg	DX Corr	Pos AD	Neg AD	% Pos AD
Linear w/MRI	0.2271	0.9195	0.2035	0.4045	0.3425	0.2178	36.2	0.2	0.2409
Linear	0.2285	0.9214	0.2063	0.3773	0.3448	0.2242	44.4	0.0	0.2482
Logistic w/MRI	0.2669	0.7101	0.1032	0.7899	0.4833	0.3665	33.8	2.6	0.3304
Logistic	0.2706	0.7042	0.1013	0.7957	0.4907	0.3619	40.6	3.8	0.3373

A small subset of patients also had CSF measurements taken at other visits including 12-month and 24-month follow-ups, but the small sample size led us to omit these particular observations.

Second Stage Initial Fits

Description

The second stage model uses ADNI3 baseline PET scans with the tracer 18 F-flortaucipir (AV-1451) and with CT scan coregistration, AC-PC orientation and standardization, intensity normalization, and variable smoothing. This is the highest degree of post-processing available in the ADNI3 dataset, and should provide the most consistency between images for each patient without performing any additional post-processing like SPM-12 mapping. There are 333 of these tau PET scans available, with 54 AD patients and 279 CN patients.

Two different models were fit here to guide further approaches: a 2D convolutional neural network (CNN) applied to a single z-axis slice for each patient, and a 3D CNN applied to a chunk of 40 z-axis slices. Each model's only task is to classify patients as 0 for CN or 1 for AD. We use a train/validate/test split of 80%/10%/10%. Each iteration or epoch of the model fit calculates the parameters for the entire training set then calculates the accuracy of these parameters using the validation set. The final parameters are then used to test the accuracy of the model on the test set using the metrics specified above. Each model uses 5 fold cross-validation, with the average prediction accuracy reported across all folds.

Code and Output

The second stage models were written in Python, and primarily utilizes the following packages: `NumPy`, `Pandas`, `PyTorch`, `NIBabel`, `MatPlotLib`, and `SciKitLearn`. The code is too long to include here in its entirety, so highlights are shown and the full code can be provided upon request.

```
# Read Data ###
dir = ('C:/Users/Ben/Desktop/Baseline/Data/')
files = os.listdir(dir)

# read in CSV description of downloaded PET scans
df = pd.read_csv('C:/Users/Ben/Downloads/Baseline_PET_All_3_20_2021.csv')

# create x and y arrays
group = []
for i in range(len(files)):
    idx = files[i][5:15]
    group.append(df.loc[df.Subject==idx,'Group'].values[0])
group = (np.array(group) == 'AD') / 1
ims = np.array([nib.load(dir + file).get_fdata() for file in files])

# create train/val/test splits
split_1 = int(len(ims) * .8)
split_2 = int(len(ims) * .9)
x, y = shuffle(ims, group, random_state = 12)

# this looks at the 30th z-axis slice. Can be changed for ensemble approaches
x_train = torch.from_numpy(np.moveaxis(x[:split_1,:,:,30], -1, 1)).float()
x_val = torch.from_numpy(np.moveaxis(x[split_1:split_2,:,:,30], -1, 1)).float()
x_test = torch.from_numpy(np.moveaxis(x[split_2:,:,:,30], -1, 1)).float()
y_train = y[:split_1]
y_val = y[split_1:split_2]
y_test = y[split_2:]

# check for proper shapes
```

```
print(x_train.shape)
print(x_val.shape)
print(x_test.shape)
print(y_train.shape)
print(y_val.shape)
print(y_test.shape)

# check for proper stratification. This is done explicity and relies on the
# random state for now.
print(sum(y_train))
print(sum(y_val))
print(sum(y_test))
```

```
torch.Size([266, 1, 160, 160])
torch.Size([33, 1, 160, 160])
torch.Size([34, 1, 160, 160])
(266,)
(33,)
(34,)
42.0
6.0
6.0
```

Here are the definitions for the 2D and 3D CNNs:

```
# https://github.com/moboehle/Pytorch-LRP/blob/master/ADNI%20Training.ipynb for reference
class ClassificationModel2D(nn.Module):

    def __init__(self, dropout=0.4, dropout2=0.4):
        nn.Module.__init__(self)
        self.Conv_1 = nn.Conv2d(1, 8, 3)
        self.Conv_1_bn = nn.BatchNorm2d(8)
        self.Conv_1_mp = nn.MaxPool2d(2)
        self.Conv_2 = nn.Conv2d(8, 16, 3)
        self.Conv_2_bn = nn.BatchNorm2d(16)
        self.Conv_2_mp = nn.MaxPool2d(3)
        self.Conv_3 = nn.Conv2d(16, 32, 3)
        self.Conv_3_bn = nn.BatchNorm2d(32)
        self.Conv_3_mp = nn.MaxPool2d(2)
        self.Conv_4 = nn.Conv2d(32, 64, 3)
        self.Conv_4_bn = nn.BatchNorm2d(64)
        self.Conv_4_mp = nn.MaxPool2d(3)
        self.dense_1 = nn.Linear(576, 128)
        self.dense_2 = nn.Linear(128, 2)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(dropout)
        self.dropout2 = nn.Dropout(dropout2)

    def forward(self, x):
        x = self.relu(self.Conv_1_bn(self.Conv_1(x)))
        x = self.Conv_1_mp(x)
        x = self.relu(self.Conv_2_bn(self.Conv_2(x)))
        x = self.Conv_2_mp(x)
```

```
        x = self.relu(self.Conv_3_bn(self.Conv_3(x)))
        x = self.Conv_3_mp(x)
        x = self.relu(self.Conv_4_bn(self.Conv_4(x)))
        x = self.Conv_4_mp(x)
        x = x.view(x.size(0), -1)
        x = self.dropout(x)
        x = self.relu(self.dense_1(x))
        x = self.dropout2(x)
        x = self.dense_2(x)
        return x

net = ClassificationModel2D()
# count parameters
sum(p.numel() for p in net.parameters())
```

98738

```
class ClassificationModel3D(nn.Module):

    def __init__(self, dropout=0.4, dropout2=0.4):
        nn.Module.__init__(self)
        self.Conv_1 = nn.Conv3d(1, 8, 3, padding = 2)
        self.Conv_1_bn = nn.BatchNorm3d(8)
        self.Conv_1_mp = nn.MaxPool3d(2)
        self.Conv_2 = nn.Conv3d(8, 16, 3, padding = 2)
        self.Conv_2_bn = nn.BatchNorm3d(16)
        self.Conv_2_mp = nn.MaxPool3d(3)
        self.Conv_3 = nn.Conv3d(16, 32, 3, padding=2)
        self.Conv_3_bn = nn.BatchNorm3d(32)
        self.Conv_3_mp = nn.MaxPool3d(2)
        self.Conv_4 = nn.Conv3d(32, 64, 3, padding=2)
        self.Conv_4_bn = nn.BatchNorm3d(64)
        self.Conv_4_mp = nn.MaxPool3d(3)
        self.dense_1 = nn.Linear(3200, 128)
        self.dense_2 = nn.Linear(128, 2)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(dropout)
        self.dropout2 = nn.Dropout(dropout2)

    def forward(self, x):
        x = self.relu(self.Conv_1_bn(self.Conv_1(x)))
        x = self.Conv_1_mp(x)
        x = self.relu(self.Conv_2_bn(self.Conv_2(x)))
        x = self.Conv_2_mp(x)
        x = self.relu(self.Conv_3_bn(self.Conv_3(x)))
        x = self.Conv_3_mp(x)
        x = self.relu(self.Conv_4_bn(self.Conv_4(x)))
        x = self.Conv_4_mp(x)
        x = x.view(x.size(0), -1)
        x = self.dropout(x)
        x = self.relu(self.dense_1(x))
        x = self.dropout2(x)
        x = self.dense_2(x)
```

```
        return x

net = ClassificationModel3D()
# count parameters
sum(p.numel() for p in net.parameters())
```

```
483138
```

Here is the code that runs the 2D CNN, with the final output. It is very similar to the corresponding code for the 3D CNN, which is intentionally omitted in the interest of space.

```
net = ClassificationModel2D()
num_epochs = 50
min_iters = 3
ignore_epochs = 15
normalize = False
retain_metric = accuracy_score
metrics = [accuracy_score]
model_path = 'C:/Users/Ben/Desktop/Collection1/torch'
r = 0

check = ModelCheckpoint(path=model_path,
                            prepend="repeat_{}".format(r),
                            store_best=True,
                            ignore_before=ignore_epochs,
                            retain_metric=retain_metric)
callbacks = [check, EarlyStopping(patience=10, ignore_before=ignore_epochs, retain_metric="loss", mode=

fold_metric, models = run(net=net, data=adni_data_train,
                  k_folds=-1,
                  callbacks=None,
                  shape=-1,
                  masked=False,
                  metrics=metrics,
                  num_epochs=num_epochs,
                  retain_metric=retain_metric,
                  b=4,
                 )

print(np.mean(fold_metric))
print(np.std(fold_metric))
```

```
All accuracies: [0.8181818181818182, 0.9090909090909091, 0.7878787878787878, 0.9090909090909091, 0.8181
0.8484848484848484
0.05070666827479244
```

We can see the average validation accuracy for the 5 folds of the 2D CNN is around 85%, which is quite high considering this only applies to a single fold.

Finally, we will look at the prediction accuracies for the CNNs. We only consider the best fold from each for simplicity, but using an ensemble approach for all 5 folds would yield very similar results since there's a high degree of consensus among test predictions for each fold.

Table 2: 2D CNN Prediction Results

	precision	recall	f1-score	support
CN	0.93	0.96	0.95	28
AD	0.80	0.67	0.73	6
—	—	—	—	—
accuracy			0.91	34
macro avg	0.87	0.82	0.84	34
weighted avg	0.91	0.91	0.91	34

Table 3: 3D CNN Prediction Results

	precision	recall	f1-score	support
CN	0.87	0.96	0.92	28
AD	0.67	0.33	0.44	6
—	—	—	—	—
accuracy			0.85	34
macro avg	0.77	0.65	0.68	34
weighted avg	0.83	0.85	0.83	34

We see that the 2D CNN is unequivocally a better classifier than the 3D CNN with this implementation.

Discussion and Follow-up

We have not yet calculated the combined model accuracy for the first stage and second stage models due to time constraints. The models for each stage were calculated independently, so there's no consistency in a test or holdout set needed for a combined accuracy. However, with the framework intact, our next step is simply to create a final holdout set, retrain the models with that data excluded, then perform all the analyses above on that set.

We see that the 2D CNN beats the best first stage model (linear without MRI) in terms of prediction accuracy, with about a 10% gain in prediction accuracy (equivalently a 10% reduction in misclassification). This means that we do not observe an increase in prediction accuracy using whole brain volume and hippocampal volume metrics in the first stage model, but we do see such an increase when only considering tau PET data for a single slice.

Given the promising results we observe for our 2D CNN, and the relative ease of computation (roughly 3 minutes for 5 folds at 60 epochs each), our next step is to turn our second stage model into an ensemble that looks at 10 z-axis slices (including the slice we already fit) and classifies using a majority vote approach. The approach for the combined model will remain the same. We will also attempt to add a third stage model that utilizes MRI scans the same way the second stage model uses tau PET scans, and include that in the combined model. We will also attempt to use layer-wise propagation techniques to visualize which voxels are most likely to lead to an AD classification within the CNN.