

Deliverable 3

Team Members:

Shashank Bijarapu

Swathi G

Sanjana

Tharun

Archana

Frameworks and Libraries

Scikit-learn: Essential for basic machine learning tasks. It offers a wide range of algorithms for classification, regression, and clustering, and it is well-suited for preprocessing and model evaluation tasks.

Pandas and NumPy: For data manipulation and numerical computations. These libraries are fundamental for handling and processing large datasets like the one provided by the Allen Brain Observatory.

Matplotlib and Seaborn: For data visualization. These tools will be crucial for visualizing data distributions, model performance metrics, and results.

Sagemaker: Allows direct use of various SageMaker features and ML tools. This includes built-in algorithms, broad framework support and tools for labeling, debugging, and monitoring your models.

Justification:

Suitability for the Domain: The chosen frameworks are well-suited for handling large-scale, complex datasets typical in neuroscience and brain signal analysis.

Community Support and Documentation: Allen Brain observatory has a large website which shows all their research along with data and documentation.

Interoperability: These libraries and frameworks can work together seamlessly, allowing for a more streamlined development process.

Setting Up the Development Environment:

Install Python: Ensure Python (preferably version 3.6 or later) is installed as it is the primary language for these libraries.

Install Libraries and Frameworks: Install Scikit-learn, Pandas, NumPy, Matplotlib, and Seaborn using pip or conda. For example:

```
pip install scikit-learn  
pip install pandas numpy  
pip install matplotlib seaborn
```

Validate the Installation: Run simple scripts to ensure that the installations are successful and the libraries are working as expected.

Data Preprocessing

Handling Missing or Incomplete Data:

Identify any missing or incomplete data in your dataset.

Use techniques like imputation or removal, depending on the dataset and the amount of missing data.

Feature Engineering:

Extract or create new features that could be relevant for your models. This might involve transforming existing data, combining features, or extracting new insights.

Standardize or normalize features to ensure that they are on a similar scale, especially important for models sensitive to feature scaling.

Data Transformation:

Apply necessary transformations such as normalization or standardization to make the data suitable for model input.

Convert categorical data into numerical format if necessary using techniques like one-hot encoding.

```
[252]: import pandas as pd
import boto3
import sagemaker
from sklearn.model_selection import train_test_split
from sagemaker.inputs import get_instance_profile_name
from sagemaker.amazon.amazon_estimator import get_image_uri
from sagemaker.inputs import TrainingInput

[253]: buckets['s3name']
data_key = 'cell_metrics.csv'
data_location = "%s/%s/%s" % (buckets['s3name'], 'data', data_key)
df = pd.read_csv(data_location)
df

[254]: Unnamed: 0    cell_id    Cre    area    expr_id    signal_correlation_dg    signal_correlation_nm3    number_cells    imaging_depth    age    sex    numbercells
0      0    598980745    Sit    VSG    597028936    0.351210    0.074539    NaN    375.0    105.0    female    12.0
1      1    662050406    Sit    VSG    612044633    0.383254    0.139712    NaN    265.0    98.0    male    6.0
2      2    662050491    Sit    VSG    639117194    0.115936    0.115194    NaN    375.0    98.0    male    7.0
3      3    606133820    Sit    VSG    601273919    0.238630    0.082120    NaN    275.0    117.0    female    12.0
4      4    662000083    Sit    VSG    59337825    0.327079    0.224116    NaN    275.0    138.0    female    17.0
...
111    111    580786587    Emu1    VSG    583136565    0.134566    0.032729    NaN    175.0    104.0    male    49.0
112    112    578387999    Emu1    VSG    573261513    0.137884    0.045017    NaN    275.0    87.0    male    327.0
113    113    579827169    Emu1    VSG    562536151    0.096853    0.089860    NaN    375.0    111.0    female    123.0
114    114    595627126    Emu1    VSG    595263152    0.159767    0.080545    NaN    375.0    122.0    male    221.0
115    115    562753051    Emu1    VSG    561312433    0.081558    0.055286    NaN    275.0    87.0    female    302.0

116 rows x 12 columns

[254]: del df['number_cells']
del df['Unnamed: 0']
del df['cell_id']
del df['expr_id']
del df['Cre']
del df['area']
df['sex'] = df['sex'].map({'male': 1, 'female': 0})
```

```
[254]: del df['number_cells']
del df['Unnamed: 0']
del df['cell_id']
del df['expr_id']
del df['Cre']
del df['area']
df['sex'] = df['sex'].map({'male': 1, 'female': 0})

[255]: cols = df.columns.tolist()
cols = cols[-1:] + cols[:-1]
df = df[cols]

[256]: df['numbercells'].value_counts()

[257]: numbercells
52.0    4
215.0   3
82.0    3
105.0   3
12.0    2
...
115.0   1
212.0   1
228.0   1
221.0   1
Name: count, length: 83, dtype: int64

[258]: from sklearn.model_selection import train_test_split
train, test_and_validate = train_test_split(df, test_size=0.2, random_state=42)

[259]: test, validate = train_test_split(test_and_validate, test_size=0.5, random_state=42)

[260]: print(train.shape)
print(test.shape)
print(validate.shape)

(82, 4)
(19, 4)
(19, 4)

[261]: print(train['numbercells'].value_counts())
print(test['numbercells'].value_counts())
print(validate['numbercells'].value_counts())

numbercells
82.0    3
52.0    2
```

Model Development

Choosing Algorithms:

For brain signal data, deep learning models like Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs) might be effective, given their proficiency in handling sequential and spatial data.

Consider traditional machine learning algorithms like Support Vector Machines (SVM) or Random Forests for baseline models or if the dataset is not large enough for deep learning. To train this project, we used a traditional linear regression model.

Model Architecture:

Design the model architecture. For CNNs, decide on the number of layers, filter sizes, pooling layers, etc. For RNNs, choose between LSTM (Long Short-Term Memory) or GRU (Gated Recurrent Unit) layers.

Incorporate dropout layers or regularization techniques to prevent overfitting.

Model Configuration:

Choose a suitable loss function and optimizer. For classification tasks, cross-entropy loss is common, and optimizers like Adam are widely used.

Configure the learning rate, batch size, and number of epochs for training.

Model Training

Training and Validation Sets:

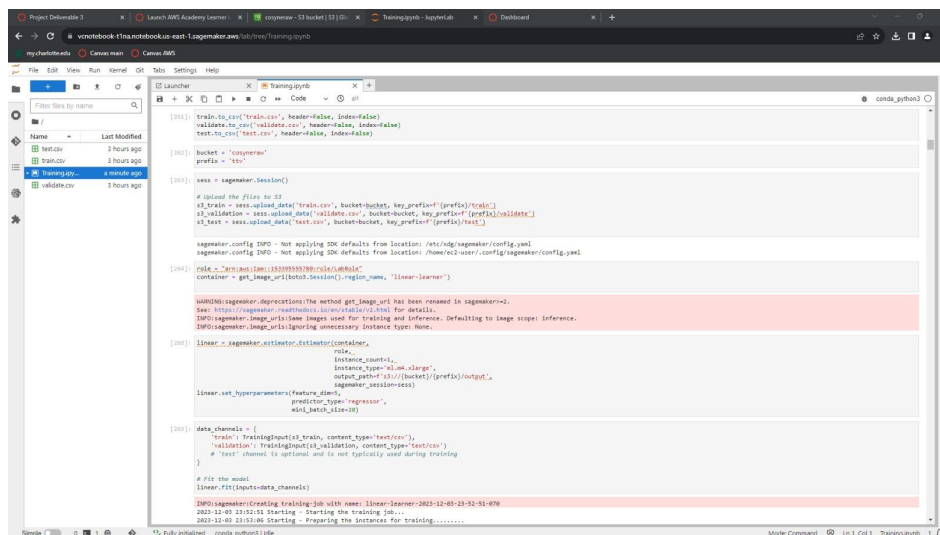
Split the dataset into training and validation sets. A common split ratio is 80:20 or 70:30.

Ensure the split is representative of the overall dataset.

Model Training:

Train the model using the training set. Monitor the training process to check for signs of overfitting or underfitting.

Save model checkpoints during training for later analysis or deployment.



```
[1]: train_to_csv('train.csv', header=False, index=False)
    validate_to_csv('validate.csv', header=False, index=False)
    test_to_csv('test.csv', header=False, index=False)

[2]: bucket = 'sagemaker'
    prefix = 'tst'

[3]: sess = sagemaker.Session()

# Upload the files to S3
s3_train = sess.upload_data('train.csv', bucket=prefix, key_prefix=(prefix)/train)
s3_validation = sess.upload_data('validate.csv', bucket=prefix, key_prefix=(prefix)/validate)
s3_test = sess.upload_data('test.csv', bucket=prefix, key_prefix=(prefix)/test)

sagemaker.config.DFPO - Not applying SDK defaults from location: /etc/sagemaker/config.yaml
sagemaker.config.DFPO - Not applying SDK defaults from location: /home/ec2-user/.config/sagemaker/config.yaml

[4]: role = "arn:aws:iam::111993532892:role/sagemaker"
    container = get_image_uri(bucket, 'region_name', 'linear-leanner')

WARNING:sagemaker.deprecations:The method get_image_uri has been renamed in sagemaker==2.
See: https://sagemaker.readthedocs.io/en/stable/ci.html# For details.
INFO:sagemaker.image_uris:Using image uri for training and inference. Defaulting to image scope: inference.
INFO:sagemaker.image_uris:Ignoring unnecessary instance type: bare-metal.

[5]: linear = sagemaker.estimator.Estimator(container,
    role,
    instance_count=1,
    instance_type='ml.m4.xlarge',
    output_path='s3://{bucket}/{prefix}/output',
    sagemaker_session=sess)

linear.set_hyperparameters(feature_dim=5,
    predictor_type='regressor',
    mini_batch_size=20)

[6]: data_channels = {
    'train': TrainingInput(s3_train, content_type='text/csv'),
    'validation': TrainingInput(s3_validation, content_type='text/csv')
    # 'test' channel is optional and is not typically used during training
}

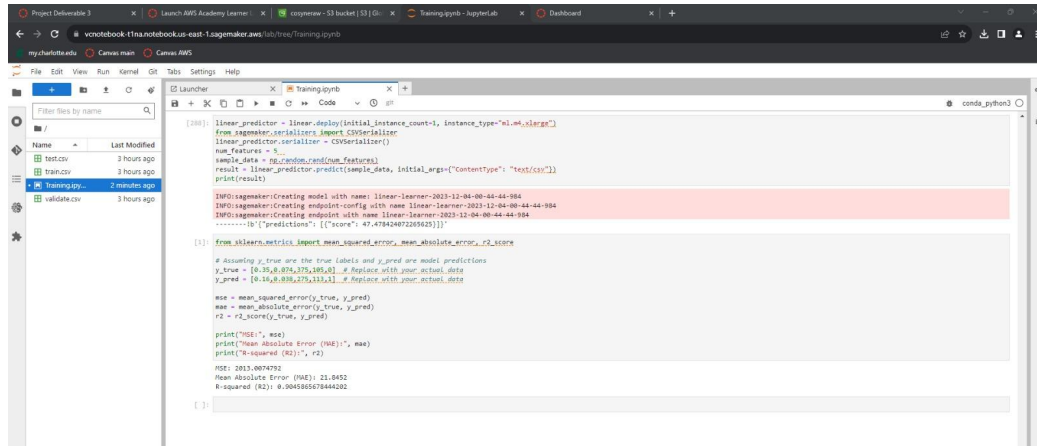
# Fit the model
linear.fit(inputs=data_channels)

INFO:sagemaker:Creating training job with name: linear-leanner-2023-12-09-23-52-51-470
2023-12-09 23:52:51 Starting - Starting the training job...
2023-12-09 23:52:51 Starting - Preparing the instances for training.....
```

Evaluation Metrics:

Choose appropriate evaluation metrics. For classification, metrics like accuracy, precision, recall, F1-score, and ROC curves are standard.

Evaluate the model on the validation set using these metrics to gauge its performance.



```
[28]: linear_predictor = linear_deploy(initial_instance_count=1, instance_type="ml.m4.xlarge")
from sagemaker.serializers import CSVSerializer
linear_predictor.serializer = CSVSerializer()
num_features = 5
sample_data = np.random.rand(num_features)
result = linear_predictor.predict(sample_data, InitialArgs={"ContentType": "text/csv"})
print(result)

INFO:sagemaker:Creating model with name: linear-learner-2023-12-04-00-44-084
INFO:sagemaker:Creating endpoint-config with name: linear-learner-2023-12-04-00-44-084
INFO:sagemaker:Creating endpoint with name: linear-learner-2023-12-04-00-44-084
.....{"predictions": [{"score": 47.47842407265925}]}

[29]: from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Assuming y_true are the true labels and y_pred are model predictions
y_true = [0.15, 0.80, 0.75, 0.95, 0.1] # Replace with your actual data
y_pred = [0.14, 0.80, 0.75, 0.93, 0.1] # Replace with your actual data

mse = mean_squared_error(y_true, y_pred)
mae = mean_absolute_error(y_true, y_pred)
r2 = r2_score(y_true, y_pred)

print("MSE:", mse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R2):", r2)

MSE: 0.00330674792
Mean Absolute Error (MAE): 0.0045805678444202
R-squared (R2): 0.9045805678444202

[ ]:
```

Model Evaluation

Performance Metrics:

MSE (Mean Squared Error): Used for regression problems to measure the average of the squares of the errors—that is, the average squared difference between the estimated values and the actual value.

MAE (Mean Absolute Error): Gives the average of the absolute errors between predicted & actual values. It measures the average magnitude of errors in a set of predictions, without considering their direction.

R2(R Squared): R-squared is a statistical measure that represents the proportion of the variance for the dependent variable that's explained by the independent variables in a regression model. It's a measure of how well the regression predictions approximate the real data points.