Hospital Forecasting Model Training and Experimentation Guide

This guide provides a description of training a model using triage data, viewing results and choosing an appropriate model to deploy for the Sagemaker endpoint. These jupyter files can be run on (ml.t2.medium Sagemaker notebook.

Directory Structure

The Sagemaker Jupyter Notebook contains the following files and folders

|----- README.md
|----- launch_training_job.ipynb
|----- Explore_Features.ipynb
|----- Visualize_Results.ipynb
|----- Train_models_cv.ipynb
|----- code/
    |----- requirements.txt
    |----- config.py
    |----- rf_script.py
    |----- xgb_script.py

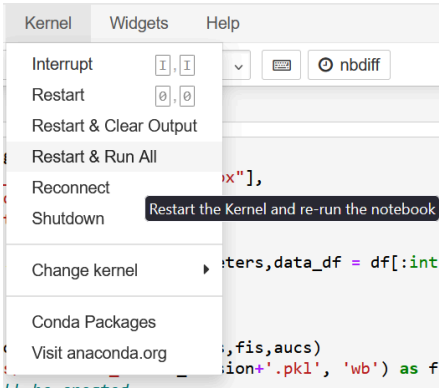
Usage Overview

Preliminary Training and Testing of Models

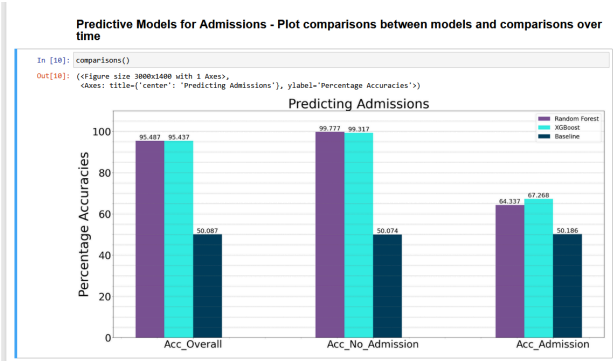
Preliminary Training, Testing and experimentation can be done via the Train_models_cv.ipynb notebook file which runs cross validation with a grid search to help the user understand the performance of models.
The cells at the end of the training notebook enable one to select which models to train along with an appropriate range of parameters to use in the Grid Search. The results of the respective models will be saved to the local results folder using the ‘test_version’ suffix to provid organization for naming different experiments.

```
In [ ]: from sklearn.ensemble import RandomForestClassifier
parameters = {'max_depth':[10],
              'n_estimators': [50,100],
              'max_features': [5,'auto']
             }
m = model_train(RandomForestClassifier(),parameters,data_df = df[:int(r*len(df))])
matrices=m[:-2]
fis=m[-2]
aucs=m[-1]
time_graphs['RF']=output_metrics(matrices,fis,aucs)
with open('./results/'+'RF'+'_'+test_version+'.pkl', 'wb') as file:
    # A new file will be created
    pickle.dump(time_graphs['RF'], file)
```

Please ensure to restart and run all cells each time in the notebook to avoid any overlap of variables:



Once models are trained and tested using cross-validation - results are stored and can be viewed using the Visualize_Results.ipynb. The following are screenshots of the different capabilities of this notebook.



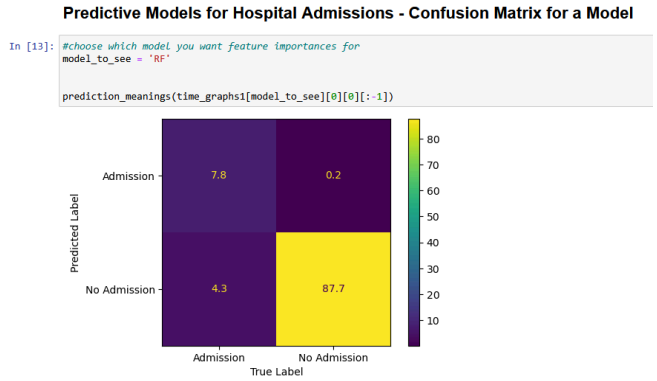
Compare Model accuracy

Display AUCS

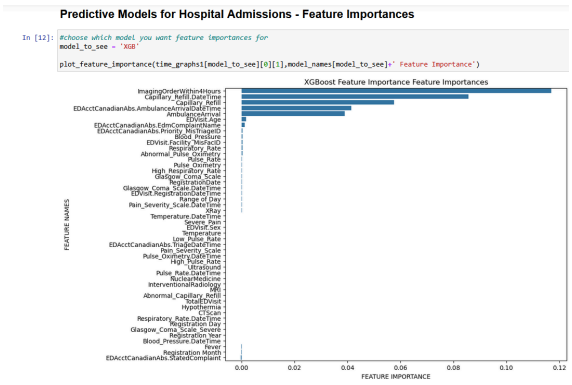
```
In [11]: from tabulate import tabulate
auc_table = []
headers = ['Model','AUC']
for i in range(len(models)):
    model = models[i]
#     print("AUC for ", model, " : ", time_graphs1[model][-1])
    auc_table.append([model,float(time_graphs1[model][-1])])


table = tabulate(auc_table, floatfmt='.3f',headers=headers)
print(table)
```

```
Model       AUC
--------  -----
RF        0.821
XGB       0.833
Baseline  0.501
```

Look at AUC Scores



Plot confusion matrices



Examine Feature Importance

Launching a training job with a finalized model

Once you have decided which model to select, you can edit the config.py file to select the final model and the final list of parameters for the model.
Once this is configured- you may run the launch_training_job.ipynb notebook to launch the training job and save the model artifact to a specified s3 directory so it can be deployed as an endpoint for inference.
The instance_type variable lets you choose a specific EC2 instance to launch the training job. It currently defaults to "ml.c5.18xlarge" for which training takes ~120 seconds.

README: Train_models_cv Notebook

Overview

This notebook handles model training with cross-validation, allowing evaluation of different models using various configurations and hyperparameters.

Configuration Parameters

The following are the key parameters used in `Train_models_cv.ipynb`:

General Settings

- **`results_folder`**: (str) Directory where the output files and trained models are stored.
- **`output_file`**: (str) Name of the output file for storing results.

- `folds`: (int) Number of cross-validation folds used to split the dataset for training and validation.

Time and Data Processing

- `all_times`: (list) A list of time windows used for evaluating models at different time intervals.
- `time_range`: (list) Specific time intervals used for generating features and performing time-based analysis.
- `window_size`: (int) The length of time windows used to process and aggregate event data.
- `timeToDeadline`: (int) Time remaining before a deadline when an event occurs, used for time-based feature extraction.

Model Hyperparameters

- `hmm_components`: (int) Number of hidden states in the Hidden Markov Model (HMM), affecting its complexity.
- `lstm_units`: (int) Number of units (neurons) in each LSTM layer, controlling model capacity.
- `learning_rate`: (float) Step size for updating model weights during optimization.
- `batch_size`: (int) Number of training samples processed before model weights are updated.
- `epochs`: (int) Number of times the model sees the entire training dataset.
- `dropout_rate`: (float) Fraction of neurons dropped during training to prevent overfitting.
- `grid_search`: (bool) Flag indicating whether to perform hyperparameter tuning using grid search.
- `cv_splits`: (int) Number of data splits used for cross-validation.
- `regularization`: (float) Strength of L1/L2 regularization applied to prevent overfitting.

Training and Evaluation

- `metrics`: (list) List of evaluation metrics used to assess model performance (e.g., accuracy, log loss, F1-score).
- `test_size`: (float) Proportion of the dataset reserved for testing.
- `train_size`: (float) Proportion of the dataset used for training.
- `random_seed`: (int) Seed value for reproducibility of experiments.
- `early_stopping`: (bool) Whether to stop training when validation performance stops improving.

- **`patience`**: (int) Number of epochs without improvement before early stopping is triggered.
- **`scoring_method`**: (str) Scoring method used for model selection during cross-validation.
- **`model_type`**: (str) Specifies the type of model being trained (e.g., HMM, LSTM, other classifiers).
- **`verbose`**: (int) Controls the level of output logging during training.

Usage

1. Run `Train_models_cv.ipynb` to perform cross-validation and evaluate different models.
2. Modify hyperparameters and configurations to experiment with different settings.
3. Review the stored results in the `results_folder` directory for further analysis.

For further customization, modify the `params` module or update configuration settings within the notebook.

README: Visualize_Results Notebook

Overview

This notebook generates visualizations and performance metrics for trained models to analyze results and compare different configurations.

Configuration Parameters

The following are the key parameters used in `Visualize_Results.ipynb`:

General Settings

- **`results_folder`**: (str) Directory where model results and performance metrics are stored.
- **`output_file`**: (str) File where processed visualization data is saved.

Visualization Preferences

- **`error_bar_type`**: (str) Defines the type of error bars displayed in plots (e.g., standard error instead of standard deviation).
- **`line_thickness`**: (int) Adjusts the thickness of plotted lines for better readability.
- **`color_palette`**: (list) Specifies colors used in plots to distinguish between different models or datasets.
- **`plot_style`**: (str) Defines the visual theme of the plots (e.g., `seaborn-darkgrid`).
- **`show_legend`**: (bool) Determines whether legends are displayed in plots.
- **`x_axis_scale`**: (str) Sets the scale of the x-axis (e.g., linear, log).
- **`y_axis_scale`**: (str) Sets the scale of the y-axis (e.g., linear, log).

Usage

1. Run `Visualize_Results.ipynb` to generate performance plots.
2. Customize visualization settings to improve interpretability.
3. Save or export figures for reports or presentations.

For further customization, modify the visualization settings directly within the notebook.

README: launch_training_job Notebook

Overview

This notebook facilitates launching model training jobs with different configurations and hyperparameters, ensuring flexible execution.

Configuration Parameters

The following are the key parameters used in `launch_training_job.ipynb`:

General Settings

- **`results_folder`**: (str) Directory where the output files and trained models are stored.
- **`output_file`**: (str) Name of the output file for storing results.
- **`num_jobs`**: (int) Number of parallel training jobs to launch.
- **`job_config`**: (dict) Dictionary containing settings for different job executions.

Model Training Settings

- **`model_types`**: (list) Specifies the types of models to train (e.g., HMM, LSTM, classifiers).
- **`hyperparameter_grid`**: (dict) Defines the range of hyperparameters for grid search optimization.
- **`train_size`**: (float) Proportion of the dataset used for training.
- **`test_size`**: (float) Proportion of the dataset reserved for testing.
- **`cv_splits`**: (int) Number of cross-validation splits.
- **`random_seed`**: (int) Seed for reproducibility.
- **`epochs`**: (int) Number of training epochs.
- **`batch_size`**: (int) Number of samples per batch during training.

Execution Settings

- **`use_gpu`**: (bool) Flag indicating whether to use GPU acceleration for training.
- **`log_level`**: (str) Controls verbosity of logging output.
- **`save_checkpoints`**: (bool) Whether to save model checkpoints during training.

Usage

1. Run `launch_training_job.ipynb` to start training models with different configurations.
2. Monitor job progress and adjust hyperparameters as needed.
3. Collect trained models and evaluate them using `Visualize_Results.ipynb`.

For further customization, modify the job configuration parameters within the notebook.