Homework #3

CSEP 590B: Explainable AI
Prof. Su-In Lee
Due: 6/1/22 11:59 PM

1 Review questions (20 points)

(a) [5 points] Linear models with a large number of features are considered less interpretable than those with a small numbers of features. Describe how having a large number of features affects the three meanings of "model interpretability" discussed in class: *simulatability*, *decomposability* and *algorithmic transparency*.

(b) [5 points] Describe the different requirements for concept labels in Concept Bottleneck Models and TCAV. Which approach requires more concept labels?

(c) [5 points] Describe the optimization procedure used to visualize what activates a neuron within a neural network (activation maximization). What are the advantages and disadvantages of this approach over visualizing the neuron using dataset examples?

(d) [5 points] Describe what *leverages scores* represent in the linear regression context. How are they calculated given a dataset $X \in \mathbb{R}^{n \times d}$ and $Y \in \mathbb{R}^n$?

2 Inherently interpretable models (45 points)

In this problem, we'll train several inherently interpretable models and see what we can learn from the results. First, download the wine quality dataset from here, and then load the dataset using the following code:

```
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv('winequality-red.csv')
X = df.drop(columns=['quality'])
y = df['quality']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42)
```

(a) [6 points] Train a standard linear regression model to predict the wine quality label, and report the root mean squared error (RMSE) on the test data. Then, show a bar plot of the coefficients for each feature. **Hint:** use `sklearn.linear_model.LinearRegression`.

(b) [8 points] It is generally a bad idea to directly interpret coefficients in a linear model, because the coefficients depend on each feature's scale. Instead, create a new linear regression model trained on *normalized* input data—that is, normalize each feature by subtracting its mean and then dividing by its standard deviation. Report the RMSE and make a bar chart of the coefficients in this model. Based on this plot, which features seem important for predicting wine quality? **Hint:** calculate the mean and standard deviation using the training data, and use these to normalize both the training and testing data.

(c) [8 points] Regularization reduces the risk of overfitting and can improve interpretability by encouraging sparsity. Fit lasso and ridge regression models using `alpha=0.1` for lasso and `alpha=10.0` for ridge, again using the normalized data from part (b). Report the RMSE for each model and make bar charts of the coefficients. **Hint:** use `sklearn.linear_model.Lasso` and `sklearn.linear_model.Ridge`.

(d) [8 points] The key hyperparameter in regularized linear regression is `alpha` (denoted as λ in the lecture slides), which controls the weight of the penalty term. Fit separate lasso regression models with the following alpha values: `[0.001, 0.01 , 0.1 , 1.]`. Then, generate three scatter plots: (1) alpha values on the x-axis and the test RMSE on the y-axis, (2) alpha values on the x-axis and the number of zero coefficients on the y-axis, and (3) number of zero coefficients on the x-axis and test RMSE on the y-axis. Consider using a log-scale for the axes where appropriate. If sparsity is viewed as a surrogate for interpretability, what does the third plot say about the accuracy-interpretability tradeoff?

(e) [8 points] Instead of assuming linear relationships between features and the output, we will now try fitting a GAM. Use `pygam` to fit a `LinearGAM` consisting of univariate splines for each feature (see here). Report the RMSE, and then create plots visualizing each univariate spline. Do the results agree with those from part (c)? Based on an internet search of how alcohol and volatile acidity affect wine quality, do your results agree with the experts? **Note:** when creating the model, use one spline term for each input variable and no factor terms.

(f) [7 points] Finally, to contextualize the previous models' accuracy, let's compare to both a strong baseline and a weak baseline. As a weak baseline, consider the simplest possible model: predicting the mean label from the training dataset. Report the test RMSE from this baseline model. Then, as a strong baseline, report the RMSE from a gradient boosting machine. How do these compare to the simpler models from the previous questions? **Hint:** use `sklearn.ensemble.GradientBoostingRegressor`.

3 Instance explanations (35 points)

In this problem, we'll implement one of the simplest instance explanations: leave-one-out. First, load a modified version of the census income (adult) dataset using the SHAP package as follows:

```
import shap
import numpy as np
from sklearn.model_selection import train_test_split


def load_mislabeled_data(test_size=0.99, n_flips=50, seed=1904):
    """Load census dataset and mislabel samples.

    Args:
      test_size: Percentage of data to reserve for test data.
        Use this parameter to shrink the size of the training data.
      n_flips: number of training labels to flip.
      seed: random seed to randomly choose samples to flip.

    Returns:
      X_train: training input data.
      X_test: testing input data.
      y_train: training output.
      y_test: testing output.
      is_flip: boolean array representing mislabeled samples.
    """
    X, y = shap.datasets.adult()

    # Train/test split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, random_state=7)

    # Randomly mislabel training data
    np.random.seed(seed)
    flip_inds = np.random.choice(
        range(X_train.shape[0]), n_flips, replace=False)
    is_flip = np.zeros(X_train.shape[0]).astype('bool')
    is_flip[flip_inds] = True
    y_train[flip_inds] = 1 - y_train[flip_inds]

    return X_train, X_test, y_train, y_test, is_flip
```

(a) [3 points] Load a *small* version of the census data with 50 mislabels and a small training dataset:

```
data = load_mislabeled_data(test_size=0.99)
```

How many training samples are in this dataset? Looking at the loading function, how are we synthetically mislabelling training data?

(b) [8 points] In this problem, our goal is to identify the mislabeled samples. We can treat this as a classification problem where we aim to rank samples according to the likelihood that they are mislabeled. We can then use metrics such as *area under the receiver operating characteristic curve* (AUROC) or *area under the precision recall curve* (AUPR) to evaluate the ordering against the mislabels, which we know because we created them (see the `is_flip` variable returned by `load_mislabeled_data`).

As a baseline, generate a random importance score for each training example by sampling from a uniform random variable. Evaluate these importance scores through their ability to identify mislabeled samples using AUROC and AUPR. In addition, generate side-by-side boxplots of the importance values for normal samples and mislabeled samples.

Hint: use `sklearn.metrics.roc_auc_score` and `sklearn.metrics.average_precision_score` for AUROC and AUPR, and `matplotlib.pyplot.boxplot` for the boxplot.

(c) [8 points] Implement the leave-one-out instance explanation approach to score each example in the training dataset. Specifically, begin by training a model f using all the data. Then, for each sample (x_i, y_i) in the training data, train a new model f_i without (x_i, y_i). The difference between the new model's test loss and the full model's test loss represents that sample's importance. For this problem, the model should be `sklearn.linear_model.LogisticRegression` and the loss should be `sklearn.metrics.log_loss`. Evaluate the leave-one-out importance scores using AUROC/AUPR and generate side-by-side boxplots of the importance scores for normal and mislabeled samples, similar to part (b).

Hint: If you encounter convergence issues with the logistic regression, try standardizing the data or increasing the `max_iter` parameter.

(d) [8 points] The leave-one-out approach should have worked fairly well in the previous setting. Now, we'll try it with larger training datasets. Report the AUROC/AUPR metrics and create boxplots for normal samples and mislabeled samples using random scores and leave-one-out scores (as in (b) and (c)) with the following two datasets:

- A *medium* dataset with 50 mislabeled examples:

  ```
  data = load_mislabeled_data(test_size=0.95)
  ```

- A *large* dataset with 50 mislabeled examples:

  ```
  data = load_mislabeled_data(test_size=0.90)
  ```

How does leave-one-out perform for these larger datasets? Why is this the case, and how could a method like Data Shapley provide better results?

(e) [8 points] Finally, one very simple method to identify mislabeled samples is to train a model on the full data and use the per-sample loss. Intuitively, bad samples are more likely to be misclassified. Implement this approach for the *large* dataset from part (d) and evaluate it as in the previous question (AUROC/AUPR and boxplots). How does this compare to the random and leave-one-out approaches?

Hint: note that `sklearn.metrics.log_loss` automatically averages the loss over all samples.