Homework 2, Due 11:59 p.m., April 21

MPCS 53111 Machine Learning

1. **Decision lists.** Exercises 18.13 and 18.14 from Russell-Norvig.

2. **Generalization Error.** Suppose I obtain, as i.i.d. samples, a training set $\mathcal{T}$ and validation set $\mathcal{V}$ for a classification task. Assume I build decision trees on the training data in which the depth is not allowed to exceed a given maximum depth value. Say, I build three such trees and measure their error rates on the validation set. Let my results be—

Max depth	Error rate
4	0.45
5	0.35
6	0.40

So for the tree with maximum depth 4, 45% of the examples in the validation dataset were misclassified by the tree. For each of the following determine if the statement is true or not, and explain why. Recall that the generalized error rate is the expected value of the misclassification error over all possible examples for the task..

- Statement 1: An unbiased estimate of the generalized error rate for a decision tree with max depth 5, trained on $\mathcal{T}$, is 0.35.
- Let an *tuned decision tree*, trained on $\mathcal{T}$, be one chosen from the three decision trees trained on $\mathcal{T}$ for which the validation error rate is minimum. (In our case, this is tree with max depth 5). Statement 2: An unbaised estimate of the generalized error rate for a tuned decision tree, trained on $\mathcal{T}$, is 0.35.
- Let a *random decision tree*, trained on $\mathcal{T}$, be one in which the max depth value is randomly chosen. Suppose the random choice happens to be 5. Statement 3: An unbaised estimate of the generalized error rate of the random decision tree is 0.35.

3. **Cross-validation**. In this problem we'll see that although cross-validation on large data sets is useful for estimating generalization error, its average-case behavior on small sets is poor and its worst-case behavior can be terrible.

Suppose that the task is binary classification and the universe U of examples has a p fraction of examples of class 1, and $1-p$ fraction of examples of class 0. Assume $0.5 < p \leq 1$ so that the majority class is 1.

Consider a simple classification algorithm A: Given a training set T of n examples, always predict the majority class in T. Concretely, if there is a class c such that T has strictly greater than $n/2$ examples of class c, predict c for every test example. Else it has to be that T has $n/2$ examples each of class 0 and class 1. In that case A randomly chooses a class $c \in \{0,1\}$ and predicts c for every test example.

Intuitively, A should have an accuracy rate of p. Let's examine how well cross-validation estimates this.

Let D be a dataset of m examples chosen i.i.d. from U. Let C denote the accuracy of A on D computed using k-fold cross-validation. C is a random variable since D is a random sample.

(a) Derive an expression for $E[C]$ in terms of p, m, and k, where the expectation is over all choices of D. Hint: Let X_i denote the class of example i in D, for $1 \leq i \leq m$. Let the training and test sets for the ith fold be T_i and S_i and let C_i be the corresponding accuracy. Derive expressions for $E[C_i|\text{majority in } T_i \text{ is } 1]$ and $E[C_i|\text{majority in } T_i \text{ is } 0]$. Subsequently derive $E[C_i]$ and $E[C]$.

(b) Using a Python script determine the value of $E[C]$ when $p = 0.7, m = 15$, and $k = 5$.

(c) Use the Python script to plot $E[C]$ for m in the range 10 to 100, and p and k as above. What can you infer from the plot?

(d) Now consider C rather than $E[C]$. Prove that as $m \to \infty$, C converges in probability to p. Hint: Use the law of large numbers.

(e) In contrast, give a worst case example where, irrespective of m or p, $C = 0$. Hint: See Exercise 18.11 in Rusell-Norvig, page 765.

4. **PAC learning.** Consider a "boolean classification" problem in which the output is boolean and the inputs are integers. Further, restrict the hypotheses to the form

$$y(\mathbf{x}) = (a \leq x_1 \leq b) \text{ and } (c \leq x_2 \leq d),$$

in which x_1, x_2 are inputs; y is the output; and a, b, c, and d are integers in $1, \dots, n$, where n is some given constant. (E.g., if $n = 100$, then one hypothesis is $(90 \leq x_1 \leq 100)$ and $(15 \leq x_2 \leq 15)$, which is true for (95, 15) and false for (95, 16).)

(a) Show that the number of hypotheses is $(n(n+1)/2)^2$. (Hint: The number of hypotheses is equal to the number of distinct rectangles on the x_1-x_2 plane with boundaries restricted to integer positions from 1 to n. Show how to count such rectangles.)

(b) How many training examples are sufficient to assure that any hypotheses with zero training error will have a generalization error of at most 10% with probability 95%.

5. **Scoring Ordinal Regression.** So far we have seen the 0/1 loss function, denoted $L_{0/1}$. Its "inverse" is accuracy—which is a *scoring function*—since the rate of $L_{0/1}$ loss is (1 − accuracy). There is a wide variety of loss/scoring functions created to address different needs.

You will implement a scoring function for the situation in which examples have to be classified into one of several rating levels, such as one to five stars. Here misclassifying a five star example as a three star should incur a higher loss than misclassifying it as a four star. $L_{0/1}$ is not appropriate as it would treat all misclassifications equally. Classifying into such rating levels is called *ordinal regression*.

A scoring function for ordinal regression is the *quadratic weighted kappa*. In the description below, we treat the "true" categories as assigned by one classifier, and the predicted categories as assigned by a second classifier. Our score then measures the similarily between the two sets of classifications, while discounting the similarity one would see simply at random—correspondng to a null hypothesis. (See Wikipedia for further explanation.)

The quadratic weighted kappa is computed by constructing three $k \times k$ matrices, where k is the number of levels or categories: O, E, and W.

- The entries o_{ij} of O are the observed proportion of observations that are classified as i by the first classifier and j by the second.

- The entries e_{ij} of E are the proportion of observations that are classified as i by the first classifier, times the proportion of observations that are classified as j by the second classifier. (These entries denote the expected proportion of joint classifications that would occur by chance.)
- The entries w_{ij} of W are the weights, which we define here to be $(i-j)^2$.

Then the quadratic weighted kappa is given by $\kappa = 1 - \frac{\sum w_{ij} o_{ij}}{\sum w_{ij} e_{ij}}$.

Implement a function `qwk(y1, y2)` that takes two classifications `y1` and `y2`, and outputs their quadratic weighted kappa. Then compare your implementation against scikit-learn's implementation[1]with random vectors to make sure yours is correct.

6. **Risk Classification for Prudential.** Here we'll explore the Prudential problem at Kaggle. This will reinforce the concepts of training, testing, crossvalidation, bias, variance, etc. Equally importantly, it will help you become familiar with scikit-learn, a popular machine learning library in Python. You will also get practice using numpy and pandas. In the following, when a function from either of the libraries is mentioned, please read the corresponding documentation to familiarize yourself with the related concepts and the input parameters. Further, whenever you need to score a model, use the **quadratic weighted kappa**—either your implementation or scikit-learn's.

   (a) Load the training data from the `train.csv` file at the site. For the remaining questions use a small enough dataset so that the plots can be constructed in a few minutes, such as a sample of 10,000 rows. Specify the number of rows you used clearly in the discussion section. (While you are developing and testing your scripts, use something even smaller.) For this you may want to use `DataFrame.sample`.

   (b) Like any real world dataset, this one is not perfect. In particular, several of the variables have missing values. Use `Imputer` to fill in the missing values. Choose an imputation strategy (mean,

[1]Use Cohen kappa with weights set to "quadratic".'

median, or most frequent) depending on the type of feature (categorical or continuous). Briefly describe your choices.

Let's say feature X_k has missing values, which you have imputed. In general, could it be useful to add an additional boolean feature $X_{k'}$ that indicates which of the rows had missing values for X_k? Explain your answer. (You are expected to give an argument here, although you may optionally support your argument by testing on this dataset.)

(c) `scikit-learn` does not allow non-numeric input variables. But for a feature with categorical values, such as "Product Info 2", we cannot simply map its values into integers, because `scikit-learn` functions assume the order between integer values is relevant. So such features are mapped into several additional variables, called dummy variables. See `OneHotEncoder` and `get_dummies`. Encode all categorical variables into dummies using either of the two functions.

(Observe that `get_dummies` has a "drop first" option. This is useful because, as we shall see when studing linear models, it helps for $\mathbf{X}$ to be full column rank.)

(d) Plot a `validation_curve` for `DecisionTreeClassifier`[2] with varying values for the `max_depth` parameter. Determine the best choice for `max_depth` from your plot. Next, with `max_depth` set to its empirically optimal value, plot the `learning_curve` for the `DecisionTreeClassifier`. Compute each score using 5-fold cross validation, by setting the corresponding parameter in the curve plotting functions.

(e) Repeat above but for `LogisticRegression`, in which vary the parameter `C`, between 1 and 1000, in the validation curve and use the optimal choice for `C` in the learning curve. Compare the learning curves for `DecisionTreeClassifier` and `LogisticRegression` and describe what you can learn from them.

Please submit (i) all your code as hw2.py and (ii) all answers to theory

[2]For both decision trees and logistic regression, use scikit-learn's implementations of the two classifiers. Further, you don't need to understand how logistic regression works—treat it like a "blackbox" classification method.

problems, all plots you obtained and your analysis of results combined as hw2.pdf.