HW5: Kernels, SVMs, and Random Forests

Last modified: 2019-04-10 16:08

Status: Released.

Due date: Fri. Apr 12 at 11:59PM EST

Turn-in links:

- PDF report turned in to: https://www.gradescope.com/courses/33142/assignments/189796/

Files to Turn In:

PDF report:

- PDF file containing neatly formatted answers to the conceptual questions below
- This document will be manually graded
- **Please**: within Gradescope via the normal submission process, annotate for each subproblem which page(s) are relevant. This will save your graders so much time!

Starter code:

See the hw5 folder of the public assignments repo for this class:

https://github.com/tufts-ml-courses/comp135-19s-assignments/tree/master/hw5

Jump to: Problem 1 Problem 2 Problem 3

Problem 1: Background

Definition: Linear kernel

A linear kernel is defined as:

$$k(x_i, x_j) = x_i^T x_j$$

where x_i and x_j are two feature vectors of the same length.

Definition: Radial basis function (RBF) kernel

A "radial basis function" (RBF) kernel is defined as:

$$k(x_i, x_j) = \exp(-\gamma \quad_{f} (x_{if} - x_{jf})^2)$$

where x_i and x_j are two feature vectors of the same length. This is also known as a "squared exponential" kernel

Definition: Kernel matrix for a dataset

Given a dataset of example feature vectors x_n, we can construct a *kernel matrix* as a $\times$ symmetric matrix where the entry at row n and column m is given by $_{n,m} = k(x_n, x_m)$.

Problem 1: Support Vector Machines and Kernels

Consider using a Support Vector Machine binary classifier in practice with an RBF kernel, such as the `SVC' implementation in sklearn.

For example, you could construct such a classifier as follows:

```
my_svm = svm.SVC(kernel='rbf', gamma=1.0, C=1.0)
```

Throughout Problem 1, you can assume a *binary* classification task with training examples, each with feature vector of size F. Each feature is a numerical real value, and each feature has been separately normalized so the training set mean is 0 and variance is 1. You can guarantee that the maximum absolute value of any feature is at most 5.

1a: Describe a setting of the hyperparameters `gamma` and `C` that is sure to *overfit* to any typical training set and achieve zero training error.

1b: Describe a setting of the hyperparameters `gamma` and `C` that is sure to *underfit* to any typical training set.

1c: If you were performing grid search to find the SVM that generalized best to unseen data, what range of values for `gamma`, `C` would you recommend? Specify your answer with two lines of NumPy code (e.g. using a list like [1, 2, 3] or `np.linspace` or `np.logspace`). Also provide 1-2 sentences of justification. Assume that you can't afford more than 5 distinct values for each variable.

```
C_grid     = __________
gamma_grid = __________
```

1d: Answer the following True/False questions, providing *one sentence* of explanation for each one

- **1d(i)**: When used for binary classification, Support Vector Machines (SVMs) provide an estimate of the probability that a given feature vector x_i will have a positive label: $p(Y_i = 1|x_i)$.
- **1d(ii)**: An advantage of an SVM is that the optimal weight vector w (a vector with one entry per feature) will typically be sparse.
- **1d(iii)**: When choosing a kernel function k, it should be the case that for any finite dataset of distinct examples, the $\times$ kernel matrix is invertible.

Problem 2: Random Forests with/without Bootstrap Aggregating (BAgging)

Consider the `RandomForestClassifier`, which is described in the ISL textbook in Chapter 8, and implemented in sklearn as described here: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

There are four key hyperparameters (using the sklearn notation):

- `n_estimators` : Number of separate trees to estimate
- `max_features` : Number of features to use at each split
- `min_samples_leaf` : Number of samples to use at each leaf
- `bootstrap` : boolean indicator of whether to perform BAgging or not

Below, you can assume a *binary* classification task with training examples, each with feature vector of size F.

2a: Describe a setting of the hyperparameters `n_estimators`, `max_features`, and `min_samples_leaf` that is sure to *overfit* to any typical training set. You can assume no BAgging (`bootstrap=False`).

2b: Describe a setting of the hyperparameters `n_estimators`, `max_features`, and `min_samples_leaf` that is sure to *underfit* to any typical training set. You can assume no BAgging (`bootstrap=False`).

2c: If you were performing grid search to find the RandomForest that generalized best to unseen data, what range of values for `n_estimators`, `max_features`, and `min_samples_leaf` would you recommend? Should we set `bootstrap` to `True` or `False`? Specify your answer with a few lines of NumPy code (e.g. using a list like [1, 2, 3] or `np.linspace` or `np.logspace` to set each variable). Include a sentence or two of justification. Assume that you can't afford more than 5 distinct values for each variable.

```
n_estimators_grid = _____
max_features_grid = _____
min_samples_leaf_grid = _____
bootstrap_grid = _______
```

2d: Answer the following True/False questions, providing *one sentence* of explanation for each one:

- **2d(i)**: When used for binary classification, RandomForests (RFs) can provide an estimate of the probability that a given feature vector x_i will have a positive label: $p(Y_i = 1|x_i)$.
- **2d(ii)**: With bootstrap aggregating enabled, random forests will almost always severely overfit the training data if the number of trees used (e.g. the `n_estimators`) is very large (say, over 500).
- **2d(iii)**: When fitting random forests, it is generally a good idea to allow each node of each tree to consider as many features as possible (e.g. `max_features` should be large).

- **2d(iv)**: Random forests only use randomness when creating many similar datasets via BAgging. No other step of the algorithm uses randomness.

Problem 3: ROC Curves Again

Previously, we've learned about ROC curves in the context of binary classifiers that produce probabilities. Here's a quick summary:

Given any dataset's true binary labels $\{y_1, y_2, \ldots y \quad\}$ and the model's predicted probabilities for each example $\{\hat{p}_1, \ldots \hat{p} \quad\}$, we can draw a ROC curve to characterize the performance of the model at all possible decision thresholds. Each point on the ROC curve gives the TPR (true positive rate) and FPR (false positive rate) at one specific threshold τ used to create binary predictions $\hat{y}_i$ at each example i:

$$\hat{y}_i = \begin{matrix} 1\text{~if~}\hat{p}_i \geq \tau \\ 0\text{~otherwise} \end{matrix}$$

- **3a:** Can we apply ROC curves to binary classifiers that cannot easily produce probabilities $\hat{p} \in (0, 1)$, but produce some real-valued scores $\hat{s} \in \mathbb{R}$ for each example? How would we select the range of thresholds to evaluate?

- **3b:** Suppose you fit a classifier to data, and you observe a TPR of 0.3 and an FPR of 0.7. Your friend says that this is worse than a random classifier (if plotted on an ROC curve, would fall below the TPR=FPR diagonal line), and so you should throw this result away. Is there anything better you can do? Describe how to tranform the predicted binary labels to reach better performance. What TPR and FPR would you expect to achieve?