CMSC 35400 / STAT 37710

Spring 2020

Homework 9

You must clearly indicate where your solutions to individual subproblems are in gradescope. If you force the graders to do this for you, points will be deducted from your total.

1. **(Kernel composition rules)** Using the basic rules for kernel composition discussed in class and assuming that $k(x, y)$ is a valid kernel, letting $f : \mathbb{R} \to \mathbb{R}$ in a) and $f : X \to \mathbb{R}$ for c) and d), and $\phi : \mathcal{X} \to \mathcal{X}'$, show that the followings are true. *(Hint: you may use the rules of addition and multiplication yielding valid kernels.)*

   a) $k_a(x, y) = f(k(x, y))$ is a valid kernel, if f is a polynomial with non-negative coefficients.

   b) $k_b(x, y) = \exp(k(x, y))$ is a valid kernel.

   c) $k_c(x, y) = f(x)k(x, y)f(y)$ is a valid kernel.

   d) $k_d(x, y) = k(\phi(x), \phi(y))$ is a valid kernel.

2. **(Kernels and feature maps)**

   a) For $x, x' \in \mathbb{R}^d$, and $k(x, x') = (x^\top x' + 1)^2$, find a feature map $\phi(x)$, such that $k(x, x') = \phi(x)^\top \phi(x')$.

   b) For the dataset $X = \{x_i\}_{i\in\{1,2\}} = \{(-3, 4), (1, 0)\}$ and the feature map $\phi(x) = [x^{(1)}, x^{(2)}, ||x||]$, calculate the kernel matrix (i.e. Gram matrix) $K = \{k(x_i, x_j)\}_{i,j\in\{1,2\}}$ (for a vector $x \in \mathbb{R}^2$ we denote by $x^{(1)}, x^{(2)}$ its components).

3. **(Gaussian Processes)** Assume $f \sim \mathcal{GP}(0, k(x, x'))$, and observation $y = f(x) + z$ where $z \sim \mathcal{N}(0, \sigma^2)$. Let $\text{var}_n(f(x^*))$ be the predictive variance of the Gaussian process regression model at x^* given a dataset of size n: $\{(x_i, y_i)\}_{i=1,\ldots,n}$. The corresponding predictive variance using a dataset of only the first $n - 1$ training points $\{(x_i, y_i)\}_{i=1,\ldots,n-1}$ is denoted $\text{var}_{n-1}(f(x^*))$.

   a) Write down the posterior variance $\text{var}_n(f(x^*))$ in terms of the Gaussian process kernel $k(\cdot, \cdot)$ and the noise parameter σ.

   b) Show that $\text{var}_n(f(x^*)) \leq \text{var}_{n-1}(f(x^*))$, i.e. that the predictive variance at x^* cannot increase as more training data is obtained.
   Hint: it may be helpful to use the matrix inversion lemma, e.g.,

$$(Z + UWV^\top)^{-1} = Z^{-1} + Z^{-1}U(W^{-1} + V^\top Z^{-1}U)^{-1}V^\top Z^{-1}$$

4. **(Programming exercise: comparison of classifers)** Consider the following classification models that we have discussed in this course:

   (1) Decision Tree
   (2) Random Forest
   (3) AdaBoost
   (4) Logistic Regression
   (5) Linear Discriminant Analysis
   (6) Naive Bayes
   (7) Neural Network
   (8) Gaussian Processes[1]
   (9) Support vector machine

   In this exercise, you will compare the performance of above models on a given binary classification task.

   Consider the synthetic dataset generated by the following python script:

```python
#python
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_classification

def create_dataset():
    X, y = make_classification(n_samples=1250,
                               n_features=2,
                               n_redundant=0,
                               n_informative=2,
                               random_state=1,
                               n_clusters_per_class=1)
    rng = np.random.RandomState(2)
    X += 3*rng.uniform(size=X.shape)
    linearly_separable = (X, y)
    X = StandardScaler().fit_transform(X)

    return X,y
```

[1] In class, we have discussed Gaussian process regression. One can also consider using Gaussian processes for classification. The key idea is to model true classification rule as being of the form

$$y = \begin{cases} 1 & f(x) \geq 0 \\ -1 & f(x) < 0 \end{cases}$$

As with regression, learning amounts to perform posterior inference, and prediction amounts to calculating the posterior mean. You may use existing GP classifier library, such as `sklearn.gaussian_process.GaussianProcessClassifier` for this task.

```
X,y = create_dataset()
```

a) Randomly split the dataset into training set X_{train} and test set X_{test}, with 80% for training and 20% for testing. Plot the decision boundaries of each of the classifiers on the test set.

You may refer to the provided code snippet `hw9_classification.ipynb` as a starting point. You can use existing machine learning packages for training different types of classifiers, e.g.

- `sklearn.neural_network.MLPClassifier`
- `sklearn.gaussian_process.GaussianProcessClassifier`
- `sklearn.ensemble.AdaBoostClassifier`, etc.

b) Now, test how the performance of each of the classifiers varies as you increase the size of the training set. Fix your test set X_{test} from step a). Starting from a (random) subset of your training set X_{train} of size 50, train your classification model. Then, increase your training set sizes to $[50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]$, by randomly adding training examples from X_{train} (with replacement). Eventually, you will select all points in X_{train}. Repeat the experiment 10 times. For each classifier, plot its average accuracy at each sample size. Compare the accuracy of different algorithms **in a single figure**, and describe your observation across different models.

For your reference, an example is provided below for the average accuracy of logistic regression on the given dataset, where the training set size increases from $[10 : 10 : 1000]$, and performance is averaged over 100 runs.

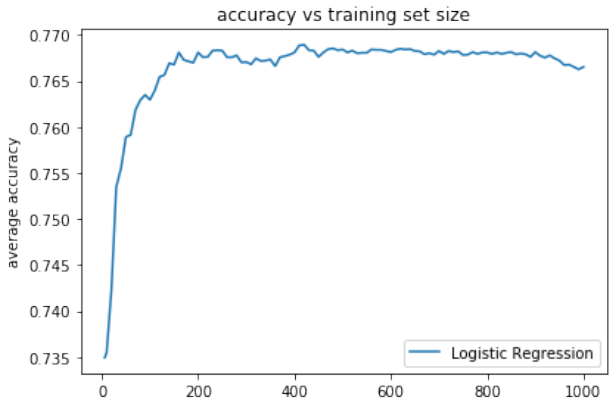

Figure 1: Example plot for average test accuracy vs sample sizes

c) Record the training time for each of the algorithms for sample sizes of $[50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]$. Plot the average training time over 10 trials of each algorithm as a function of sample sizes. Qualitatively explain how different algorithms scale with the data set sizes.