CSCE 478/878 Recitation 4 Handout
Investigation on The K Nearest Neighbor Model
January 29, 2019

Introduction

For this recitation you will investigate the following:
- Impact of scaling
- Minkowski vs. Mahalanobis distance metric
- Understanding covariance

As with last week, we will be working in and submitting Jupyter notebooks.

A couple general instructions before we start:

- Your Jupyter notebook should be submitted via webhandin by 4:45PM, **February 1**.
    - Use the same naming convention as last week: `<lastname>_<firstname>_2.ipynb`
- Use markdown cells with **bold underlined** and a font size of 6 titles to denote the start of each problem, e.g.

Jupyter Notebook Pointers

Here are a couple tips to (maybe) make your life easier:

- You can run the currently selected cell with `ctrl + enter`
    - `shift + enter` will both run the currently cell and make a new one
- Tab completion is available in many circumstances
- If you want to clear memory and start over, use the 'Kernel' dropdown menu at the top and select 'Restart'
    - 'Restart and Run All' is a great way to make sure your code works when run from scratch
- You can change the cell type between code and markdown using the 'Cell' dropdown menu
- This article contains several other useful shortcuts and tips
- This article contains info on some of the most common "magic commands" In Jupyter notebooks
    - %system pwd is a nice way of reminding yourself where your notebook is running
    - %matplotlib inline is necessary for displaying your matplotlib pictures in your notebook

Part 1: K-NN Classification Using Python (75 pts)

1. Use the "Synthetic 3D Dataset Generation Code.py" file to create your dataset for this recitation. Perform k-NN binary classification. **Don't scale the data**. **Generate the dataset five times** and do the classification five times by using the **Minkowski distance metric**.Then, report the optimal hyperparameter values (obtained from grid search cross-validation) and the following performance measures on test data in the following table.

	weights	n_neighbors	p	Accuracy	Precision	Recall	F1 Score
1							
2							
3							
4							
5							

2. Scale the data and repeat the steps from above. Generate a similar table.
3. For this step don't scale the data. Restart your jupyter notebook. **Generate the dataset five times** and do the classification 5 times by using the **Mahalanobis distance metric**. Then, report the optimal hyperparameter values (obtained from grid search cross-validation) and the performance measures on test data in the following table.

	weights	n_neighbors	Accuracy	Precision	Recall	F1 Score
1						
2						
3						
4						
5						

Note: to use the Mahalanobis distance metric, you need to compute the covariance and the inverse of the covariance of your train dataset. You may use the inv() function from the numpy.linalg library.
https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.linalg.inv.html

The inv() function might give you error if your covariance matrix is singular. In that case you need to use the pseudo inverse function pinv().
https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.linalg.pinv.html

With the Mahalanobis distance metric, you cannot tune the hyperparameters using sklearn's grid search cross-validation.

Because for each iteration of the cross-validation, a new train fold set is created which requires to compute the inverse of its covariance matrix.

The sklearn grid search does not compute the inverse of the covariance matrix automatically for each train fold set.

Thus, unfortunately, we can't experiment (tune hyperparameters) with the Mahalanobis distance metric using the grid search.

Also note that with the Mahalanobis distance metric only the brute force algorithm works. You will have to do this manually (vary the hyperparameters and evaluate the model to find the optimal parameter values).

Following code snippet shows how to use the Mahalanobis distance metric to train the sklearn k-NN model.

knn = KNeighborsClassifier(weights="uniform", algorithm='brute', n_neighbors=1, metric = "mahalanobis", metric_params={'V': covarianceMatrix})

You need to vary the following two hyperparameters to tune the model: "weights" & "n_neighbors"

The "metric_params" attribute is used to provide the covariance matrix.

4. Which distance metric gave you better performance? Justify your answer.

Part 2: Theoretical Analysis of Data Covariance (25 pts)

1. Compute the covariance matrix of the **white wine dataset** from last week. Say that it has ***n*** data points and each data point is ***d***-dimensional (excluding the target). What is the dimension of the covariance matrix?
2. Compute the inverse of the above covariance matrix using python (the inv() function of NumPy's linalg library).
3. In general, why do you think a covariance matrix could be singular? For example, consider the following matrix and compute its covariance matrix. Then, answer questions from 3 – 6.

$$\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$$

4. Is a singular matrix positive definite/semi-definite? Justify your answer.

5. Say that we can't use the pseudo inverse function to compute the inverse of a singular covariance matrix. Then, how do we fix the singularity problem? Hint: we need to edit the data. But how?
6. Determine the computational complexity of inverting a ***d*** x ***d*** matrix.

To answer the above questions, you may refer to following set of slides: "*ML-Linear Algebra for Machine Learning-3*" and "*ML-Linear Algebra for Machine Learning-5*".