Hayden Kirkeide

CAP 5771

Professor Dong

30 November 2025

Assignment #2

Using Python within the Google CoLab environments, I completed Problems 1, 2, and 3 for Assignment #2. For problem 1, I first began by referencing the class notes for Support Vector Machines to determine what exactly the algorithm was requiring us to do. After reviewing the notes, I learned that the polynomial kernel was sort of a catch-all, allowing the algorithm to have a certain degree of freedom (not being an indicator of statistical degrees of freedom) when grouping data. Kernels are utilized when nonlinear data – such as the content of images belonging to multiple different landscapes or classes, as the dataset for Problem 1 was – is presented; from my additional research, polynomial kernels are used in a more global context where Gaussian kernels are utilized more frequently in local contexts (i.e. gauging similarity/boundaries between neighbors). The accuracy values of the polynomial and gaussian kernels calculated were 62.33% and 63.43% when the gamma value is scaled, respectively.

Both of these accuracy results align with the goal of the assignment because essentially what the two kernels were doing is comparing the multiple classes present in the image within both global versus local contexts. For example, if one image contains a beach sunset view with mountains in the background and another has a forest sunset, the Gaussian kernel would have more success due to the higher likelihood of similarity to the images around it (i.e. the images would be less similar compared to the images globally but they would be more similar to each other because they contain more common "classes" and would therefore be closer together). In

short, polynomial accuracy rates are likely to be lower due to the global scope whereas Gaussian accuracy rates are likely to be higher due to the close proximity of images as a result of class presence.

For problem 2, the first step I did was import the data into Excel to gauge the values I was working with. Following my initial data exploration, I performed a preliminary analysis using a k-means Python script using the built-in function from the SciKit library. Afterwards, I found some sources for coding a k-means algorithm that did not require the k-means plug-in and would still output the required SSE values. The specific run of rounded SSE detailed in this report for k = 3 was 588.1, k = 5 was 407.8 and k = 7 was 303.8. As the "seeds.txt" dataset is quite large, it makes sense for the SSE values to be larger than expected due to the large amount of samples within it. What is expressed in the SSE values and is to be expected is the promising result of the SSE decreasing as the number of centroids increases. As k-values increase, the distance between the values decreases and thus the SSE decreases, indicating a step in the right direction.

For problem 3, I again imported the data into Excel to view the data I was working with. Contrary to the first two datasets, the data contained alphanumeric values and not strictly numeric but the dataset documentation was helpful in understanding the values present. After reading a few papers but most importantly *Balancing the Scales: A Comprehensive Study on Tackling Class Imbalance in Binary Classification* by Mohamed Abdelhamid and Abhyuday Desai, I was introduced to the Synthetic Minority Over-sampling Technique, also known as SMOTE. After conducting further research into the SMOTE libraries, I decided that – as the title of the paper implies – the SMOTE algorithms would help address the class imbalance within the dataset. All methods of the problem utilized random forest classification to randomly select and

group card values while using both weighted and unweighted (macro) metrics to accurately view the F1 scores for all methods used on the dataset. Method #1 of the exploration evaluated the F1 score metrics from a baseline where the macro (again, unweighted) score for method #1 to obtain a baseline score resulted in an F1 score of 0.7196 while the weighted score was a 0.7731. As expected, the weighted score which slightly aids in addressing class imbalances was higher (if marginally) and thus indicated a more accurate goodness of fit, resulting in a positive result for the first method. Method #2 relied on undersampling – or training an algorithm on one dataset and applying it to another without training it on the new one to account for overrepresentation – to yield an F1 score. Yet again using a random forest tree model, 70% of the samples were randomly grouped for training the model while the remaining 30% were tested. The unweighted undersampling yielded an F1 score of 0.6806 – the lowest F1 score of the methods tested, confirming its limitations – while the weighted undersampling method yielded that of a 0.7121, only marginally distant from the unweighted baseline sampling method. Finally, method #3 utilized the aforementioned SMOTE method from the Imbalance Learn library. As it is a binary classification function, its use was perfect for this training exercise. Once again using the random forest method to randomly select and group samples, the F1 score for the macro method was that of a 0.6933 – only marginally better than the unweighted undersampling method and once again showcasing its limitations – while the F1 score for the weighted method was a 0.7460, a result of oversampling the class with lower frequency yet it yields the best score of any imbalance adjustments thus far. What this observation tells me is that while it seems to be counterintuitive, “overtraining” the classification algorithm on the minority class (i.e. “overtelling” it what to look resulting from a lack of natural class occurrence in the data” yields the best results, a sign that being overly cautious may be fruitful in the case of machine learning.