Lab 4 k-Nearest Neighbors

CS3400 Machine Learning

Learning Outcomes

- Sketch or plot the decision boundaries for fitted models with 1 or 2 features
- Describe how the decision boundaries for a linear model differ from a non-linear model
- Describe cross-fold validation and what types of problems for which they are appropriate
- Interpret a metric to assess the quality of a model's predictions and choose between two models (model selection)
- Calculate the distance between two data points using a given metric
- Describe and implement the algorithm for predicting values using k-nearest neighbors
- Explain when it is appropriate to use mode, average, and median aggregation functions
- Describe the run-time of fitting and predicting with kNN in big-o notation
- Describe the limitations of kNN

Overview

In the last lab, you explored linear decision boundaries. In that lab, we gave you an equation for linear decision boundaries. We asked you to classify data points using linear decision boundaries we provided and evaluate the accuracy of those predictions. In this lab, you're going to interrogate the decision boundary learned by a machine learning model.

In many data sets, the classes cannot be separated by a line. We need to turn to models capable of learning non-linear decision boundaries. K-Nearest Neighbors (kNN) is a simple, non-parametric[1] machine learning method capable of learning non-linear decision boundaries. kNN was one of the first machine learning algorithms developed. kNN uses the "features" to represent the data as points in a N-dimensional space. kNN then calculates the distance between the query point and every training point. kNN uses the distances to find the k nearest training points and predict the value of dependent variable for the query point by calculating the mode or average of the neighboring points' dependent variable values. While it is not widely used outside of recommendation systems[2] (e.g.,) these days, it is a good introduction to implementing machine learning algorithms and counterexample for linear models.

You are going to implement the kNN algorithm for classification and regression. You will then perform two experiments with your implementation. First, you will evaluate the model on a grid

[1] Non-parametric means that the model does not assume any particular form and is completely derived from the data. For example, a histogram is a non-parametric distribution while a normal distribution is parametric.

[2] For example, a system that recommend a movie to watch based on movies you rated.

of points to visualize the learned decision boundaries. Secondly, you will explore how the parameter k impacts the accuracy and decision boundaries of the model.

In kNN, we have two sets of points: the reference points (training data) and query points (naïve data / data you want to classify). Each set of points is represented by a matrix of features. For the reference points, we have known values for the output variable which are stored as a vector. The algorithm takes a parameter k, which indicates how many reference points to use when predicting the output variable for each query point. The classification / regression algorithm proceeds as follows:

For each query point q:

1. Compute the Euclidean or squared Euclidean distance between q and every reference (training) point (Hint: use scipy.spatial.cdist)
2. Find the k closest reference points to point q (Hint: use numpy.argsort)
3. Predict the outcome variable
    a. If used for classification, predict the output variable's value as the mode of the output values for the k-nearest neighbors (Hint: use scipy.stats.mode)
    b. If used for regression, predict the output variable's value as the average of the output values for the k-nearest neighbors (Hint: use numpy.mean)

Instructions

Implement the kNN Algorithm

You have been provided with two .py files (knn_stub.py, test_knn.py). The file knn_stub.py contains the skeleton of a KNN class. Your job is to finish that implementation. The file test_knn.py contains unit tests designed to help you check the correctness of each method.

1. Rename knn_stub.py to knn.py.

2. Implement the following methods:

- __init__(self, k, aggregation_function)
- fit(X, y)
- predict(X)

You can test your code by running the following in a notebook:

```
!python test_knn.py
```

✓

Explore Decision Boundaries for 3 Data Sets

✓ 1. Create a Jupyter notebook named lastname_lab03. The notebook should have a title, your name, and an introduction.

✓ 2. In the notebook, import your KNN class.

✓ 3. For each of the three provided data sets (moon, circles, and rocky ridge):

✓ a. split the data set into training and testing sets using the `sklearn.model_selection.train_test_split()` function.

✓ b. Train the `sklearn.preprocessing.StandardScaler` transformer on the training set. Use it to transform the features for the training and testing sets.

```
scaler = StandardScaler()
train_X = scaler.fit_transform(train_X)
test_X = scaler.transform(test_X)
```

✓ c. Fit a KNN model with 3 neighbors on the training data set using the appropriate aggregation function.

✓ d. Evaluate the model for each point on a 2D grid. The grid should span from -3 to 3 on the x axis and -3 to 3 on the y axis with a point every 0.02 units.

✓ e. Plot the predicted values using `matplotlib.pyplot.contourf()`. On the same plot, create a scatter plot of data points colored by class.

Choosing an Optimal Value for k

✓ 1. Load the fourth data set (sweep.csv).

2.Evaluate the impact of the k parameter using cross-fold validation. Repeat the following process for values of k from 1 to 200 in steps of 10.

✓ a. Use `sklearn.model_selection.StratifiedKFold(n_splits=10, shuffle=True)` to divide the data set into 10 folds.

✓ b. For each fold, train the model on the remaining 9 folds and make predictions for the held-out fold.

✓ c. Calculate the accuracy of the predictions for each fold.

✓

d. Calculate the average (using `numpy.mean`) and standard deviation (using `numpy.stdev`) of the 10 accuracy scores.

✓ 3. Use `matplotlib.pyplot.errorbar()` to plot the average and standard deviation (as error bars) for each value of k.

Reflection Questions

KNN Implementation (Problem 1)

1. Estimate the run time complexity in big-o notation of training a KNN model. Justify your answer.

2. Estimate the run time complexity in big-o notation of predicting the output value for a query point using a training set of N points, with p features, and k neighbors with a KNN model. Justify your answer.

3. What do you think the potential downsides of the k nearest neighbors algorithm are? Why do you think it might not be used as widely as other methods?

Decision Boundaries (Problem 2)

1. For each of the three data sets, do you think a linear decision boundary could be used to accurately classify the data points?

2. What do we mean by a "non-linear" decision boundary? Give an example of a non-linear function that could be used as a decision boundary for one of the data sets.

3. What are the advantages of non-linear decision boundaries over linear decision boundaries? What are the disadvantages?

Choosing an Optimal Value for k (Problem 3)

1. What value of k gave the highest accuracy?

2. For large values of k, what happened to the accuracy? Why do you think this is?

3. Let's say that we ask you to use the following experimental setup to find a value of k that maximizes the accuracy of the model's predictions: split the data set into training and testing sets, train a model for each value of k using the training set, predictions the classes of the testing points, and evaluate the accuracy of the predictions. Could this approach give you misleading results? If so, why?

4. It is considered a "best practice" to use cross-fold validation when optimizing parameters. Why do you think that is?

Submission Instructions and Grading Criteria

Save the notebook as a PDF named lastname_lab04.pdf. Put your PDF and knn.py into a zip file. Upload the zip file to Canvas. Print off and submit a hard copy of this pdf during the following week's lab meeting.

We will be will looking for the following:

- That all of the unit tests pass
- That you used Numpy functions as much as possible and avoided writing loops
- An introduction (including your own statement of the problem) and a written summary of your results at the top of the notebook in Markdown. Make sure to put your name at the top of the notebook.
- That your plots look reasonable.
- Reasonable attempts to answers the questions.

Rubric

Presentation	
Followed submission and formatting instructions	5% ✓
Plots: axes are properly labeled, used correct axes for variables, points were colored as required, lines were coloring as required, used a legend, chose appropriate axes limits to make plot readable and do not cause misleading interpretations, etc.	5% ✓
KNN Implementation	✓
Implementation passes unit tests	20% ✓
Used Numpy functions wherever possible (no Python loops)	5% ✓
Decision Boundary Plots	
Correctly used train-test splitting	5% ✓
Correctly evaluated predictions on the grid	5% ✓
Accuracy: 3 total plots, contours and points are where they are supposed to be	5% ✓
Choosing an Optimal Value for k	
Correctly used cross-fold validation	5% ✓
Correctly evaluated the average and variance of the accuracy for each value of k	5% ✓
Plot of accuracy vs k is accurate	5% ✓
Reflection Questions	
Problem 1	10% ✓
Problem 2	10% ✓
Problem 3	10% ✓
Exceeded Expectations	5% ✓