Mid Term

Date: 16/01/2021

Name: Polin Rahman

Id: 1531377042

Course: CSE 445

Sec: 03

Table of Contents

Question-1

Curse of Dimensionality:

Curse of Dimensionality refers to a set of problems that arise when working with high-dimensional data. The dimension of a dataset corresponds to the number of attributes/features that exist in a dataset. A dataset with a large number of attributes, generally of the order of a hundred or more, is referred to as high dimensional data. Some of the difficulties that come with high dimensional data manifest during analyzing or visualizing the data to identify patterns, and some manifest while training machine learning models. The difficulties related to training machine learning models due to high dimensional data is referred to as 'Curse of Dimensionality'. The popular aspects of the curse of dimensionality; 'data sparsity' and 'distance concentration'.

The curse of dimensionality basically means that the error increases with the increase in the number of features. It refers to the fact that algorithms are harder to design in high dimensions and often have a running time exponential in the dimensions. A higher number of dimensions theoretically allow more information to be stored, but practically it rarely helps due to the higher possibility of noise and redundancy in the real-world data.

Regularization can help us to get rid of overfitting:

In supervised machine learning, models are trained on a subset of data aka training data. The goal is to compute the target of each training example from the training data.

Now, overfitting happens when the model learns signal as well as noise in the training data and wouldn't perform well on new data on which the model wasn't trained on.

There are few ways to avoid overfitting your model on training data like cross-validation sampling, reducing the number of features, pruning, regularization, etc.

Regularization basically adds the penalty as model complexity increases. Regularization parameter (lambda) penalizes all the parameters except intercept so that model generalizes the data and won't overfit.

Regularization, significantly reduces the variance of the model, without a substantial increase in its bias. So, it all comes down to the tuning parameter λ, which controls the impact of bias and variance. As the value of λ increases, it reduces the coefficients thus reducing the variance.

Question-2

One-hot encoding:

For categorical variables where no such ordinal relationship exists, the integer encoding is not enough.

In fact, using this encoding and allowing the model to assume a natural ordering between categories may result in poor performance or unexpected results (predictions halfway between categories).

In this case, a one-hot encoding can be applied to the integer representation. This is where the integer encoded variable is removed and a new binary variable is added for each unique integer value.

In the “color” variable example, there are 3 categories and therefore 3 binary variables are needed. A “1” value is placed in the binary variable for the color and “0” values for the other colors.

For example:

red,	green,	blue
1,	0,	0
0,	1,	0
0,	0,	1

The binary variables are often called “dummy variables” in other fields, such as statistics.

Feature Scaling:

The real-world dataset contains features that highly vary in magnitudes, units, and range. Normalization should be performed when the scale of a feature is irrelevant or misleading and not should Normalise when the scale is meaningful.

The algorithms which use Euclidean Distance measure are sensitive to Magnitudes. Here feature scaling helps to weigh all the features equally.

Formally, If a feature in the dataset is big in scale compared to others then in algorithms where Euclidean distance is measured this big scaled feature becomes dominating and needs to be normalized.

Examples of Algorithms where Feature Scaling matters

1. K-Means uses the Euclidean distance measure here feature scaling matters.

2. K-Nearest-Neighbours also require feature scaling.

3. Principal Component Analysis (PCA): Tries to get the feature with maximum variance, here too feature scaling is required.

4. Gradient Descent: Calculation speed increase as Theta calculation becomes faster after feature scaling.

Feature Scaling Example (code segment):

```
import pandas as pd

from sklearn.preprocessing import StandardScaler

# Read Data from CSV

data = read_csv('example.csv')

data.head()

# Initialise the Scaler

scaler = StandardScaler()

# To scale data

scaler.fit(data)
```

Question-3

Principal components:

Principal components are the coordinates of the observations on the basis of the new variables (namely the columns) and they are the rows of. The components are orthogonal and their lengths are the singular values. In the same way, the principal axes are defined as the rows of the matrix.

Difference between the first and the last principal component:

PC1 will always be larger than PC2, which will always be larger than PC3, and so on. In a very hand-wavy way, PC1 is the way that all your data points are similar to each other, while PC2 is one of the ways that they're all different from one another.

KNN algorithms work better on small data sets:

K Nearest Neighbor is one of the fundamental algorithms in machine learning. Machine learning models use a set of input values to predict output values. KNN is one of the simplest forms of machine learning algorithms mostly used for classification. It classifies the data point on how its neighbor is classified.

When need to use KNN:

a. We have properly labeled data. For example, if we are predicting someone is having diabetes or not the final label can be 1 or 0. It cannot be NaN or -1.

b. Data is noise-free. For the diabetes data set, we cannot have a Glucose level as 0 or 10000. It's practically impossible.

c. Small dataset.

So can say that KNN algorithm is a good choice if have a small dataset and the data is noise-free and labeled. When the data set is small, the classifier completes execution in a shorter time duration. If your dataset is large, then KNN, without any hacks, is of no use.

Question-4

F1 score :

F1 score is used as a performance metric for classification algorithms.

Firstly we need to know about the confusion matrix. Confusion matrix comes into the picture when you have already build your model.

This is what a confusion matrix looks like.

Predicted is the values that have been predicted by the model on the validation set. Actual is the ground truth available to you in the validation set.

		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

Formally;

$$Precision = \frac{TP}{(TP+FP)}$$

$$Recall = \frac{TP}{(TP+FN)}$$

To make a mental model for precision and recall, think of it in terms of recommending something to a user.

The Precision is the metric which tells out of all the recommendation that gave, how many items did the user actually like.

The Recall is the metric that tells out everything that a user would have liked, how much did you actually recommend.

Now coming to the F1 score, it's the harmonic mean of precision and recall.

$$F1 = \frac{1}{(Precision^{-1} + Recall^{-1})}$$

$$F1 = \frac{1}{\frac{1}{Precision} + \frac{1}{Recall}}$$

$$F1 = 2 * \frac{(Precision * Recall)}{(Precision + Recall)}$$

F1 score is heavily used as a performance metric in Information Retrieval tasks such as search, document classification, or any task where needs a high True Positive score. It can range from 0 to 1.

Handling missing or corrupted data in a dataset:

1. Deleting Rows

This method commonly used to handle null values. Here, we either delete a particular row if it has a null value for a particular feature and a particular column if it has more than 70-75% of missing values. This method is advised only when there are enough samples in the data set. One has to make sure that after we have deleted the data, there is no addition of bias. Removing the data will lead to loss of information which will not give the expected results while predicting the output.

Pros:

- Complete removal of data with missing values results in a robust and highly accurate model
- Deleting a particular row or a column with no specific information is better since it does not have a high weightage

Cons:

- Loss of information and data
- Works poorly if the percentage of missing values is high (say 30%), compared to the whole dataset

2. Replacing With Mean/Median/Mode

This strategy can be applied to a feature that has numeric data like the age of a person or the ticket fare. We can calculate the mean, median, or mode of the feature and replace it with the missing values. This is an approximation that can add variance to the data set. But the loss of the data can be negated by this method which yields better results compared to the removal of rows and columns. Replacing the above three approximations is a statistical approach to handling the missing values. This method is also called leaking the data while training. Another way is to approximate it with the deviation of neighboring values. This works better if the data is linear.

Pros:

- This is a better approach when the data size is small
- It can prevent data loss which results in the removal of the rows and columns

Cons:

- Imputing the approximations add variance and bias
- Works poorly compared to another multiple-imputations method

3. Assigning An Unique Category

A categorical feature will have a definite number of possibilities, such as gender, for example. Since they have a definite number of classes, we can assign another class for the missing values. Here, the features Cabin and Embarked have missing values which can be replaced with a new category, say, U for ‘unknown’. This strategy will add more information to the dataset which will result in the change of variance. Since they are categorical, we need to find one-hot encoding to convert it to a numeric form for the algorithm to understand it. Let us look at how it can be done in Python:

Pros:

- Fewer possibilities with one extra category, resulting in low variance after one hot encoding — since it is categorical
- Negates the loss of data by adding a unique category

Cons:

- Adds less variance
- Adds another feature to the model while encoding, which may result in poor performance

4. Predicting The Missing Values

Using the features which do not have missing values, we can predict the nulls with the help of a machine learning algorithm. This method may result in better accuracy unless a missing value is expected to have a very high variance.

Pros:

- Imputing the missing variable is an improvement as long as the bias from the same is smaller than the omitted variable bias
- Yields unbiased estimates of the model parameters

Cons:

- Bias also arises when an incomplete conditioning set is used for a categorical variable
- Considered only as a proxy for the true values

5. Using Algorithms Which Support Missing Values

KNN is a machine learning algorithm that works on the principle of distance measure. This algorithm can be used when there are nulls present in the dataset. While the algorithm is applied, KNN considers the missing values by taking the majority of the K nearest values. In this particular dataset, taking into account the person's age, sex, class, etc, we will assume that people having the same data for the above-mentioned features will have the same kind of fare.

Unfortunately, the SciKit Learn library for the K – Nearest Neighbour algorithm in Python does not support the presence of the missing values.

Another algorithm that can be used here is RandomForest. This model produces a robust result because it works well on non-linear and categorical data. It adapts to the data structure taking into consideration the high variance or the bias, producing better results on large datasets.

Pros:

- Does not require the creation of a predictive model for each attribute with missing data in the dataset
- Correlation of the data is neglected

Cons:

- Is a very time-consuming process and it can be critical in data mining where large databases are being extracted
- Choice of distance functions can be Euclidean, Manhattan, etc. which is do not yield a robust result

Question-5

Here is the screenshot of my output, also upload the ***.ipynb*** file individually.

Spliting dataset into 80% traindata and 20% test data

In [6]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

Training set contains 120 records

In [7]:
```
print("X_train shape: {}".format(X_train.shape))
print("Y_train shape: {}".format(y_train.shape))
```

```
X_train shape: (120, 4)
Y_train shape: (120,)
```

Test set contains 30 records

In [8]:
```
print("X_test shape: {}".format(X_test.shape))
print("Y_test shape: {}".format(y_test.shape))
```

```
X_test shape: (30, 4)
Y_test shape: (30,)
```

Train the model and calculate the accuracy of the model using the K Nearest Neighbor Algorithm

In [9]:
```
# Fitting clasifier to the Training set
# Loading libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import cross_val_score

# Instantiate learning model (k = 3)
classifier = KNeighborsClassifier(n_neighbors=3)

# Fitting the model
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
```

In [10]:
```
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[10]:
```
array([[11,  0,  0],
       [ 0, 12,  1],
       [ 0,  0,  6]], dtype=int64)
```

In [11]:
```
accuracy = accuracy_score(y_test, y_pred)*100
print('Accuracy of our model is equal ' + str(round(accuracy, 2)) + ' %.')
```

```
Accuracy of our model is equal 96.67 %.
```