Final Exam

Date: 23/01/2021

Name:	**Polin Rahman**
Id:	**1531377042**
Course:	**CSE 445**
Sec:	**03**

Table of Contents

Question - 1

Regularization is needed because

The idea behind L1 regularization is to make reduce the dataset to only the most important features that would impact the “target variable”. By adding the above-mentioned penalty, some of the coefficients of the features become 0 and the remaining features will be the most useful ones.

This is also a form of regression, that constrains/regularizes or shrinks the coefficient estimates towards zero. In other words, this technique discourages learning a more complex or flexible model, so as to avoid the risk of overfitting. A simple relation for linear regression looks like this.

On the other hand, PCA is just for Dimensionality reduction involves reducing the number of input variables or columns in modeling data. PCA is a technique from linear algebra so that can be used to automatically perform dimensionality reduction.

Question - 2

Lasso Regression:

The acronym “LASSO” stands for Least Absolute Shrinkage and Selection Operator.

Lasso regression is a type of linear regression that uses shrinkage. Shrinkage is where data values are shrunk towards a central point as they mean. The Lasso procedure encourages simple, sparse models. This particular type of regression is well-suited for models showing high levels of multicollinearity or when want to automate certain parts of model selection, like variable selection/parameter elimination.

Ridge Regression:

Ridge regression is a way to create a parsimonious model when the number of predictor variables in a set exceeds the number of observations, or when a data set has multicollinearity. Tikhivov’s method is basically the same as ridge regression, except that Tikhonov’s has a larger set. It can produce solutions even when your data set contains a lot of statistical noise.

Difference between LASSO and Ridge Regression:

Lasso and Ridge’s regression use two different penalty functions.

- Lasso uses L1, Whereas Ridge uses L2.
- In Lasso regression, the penalty is the sum of the absolute values of the coefficients. It’s a shrinkage towards zero using an absolute value rather than a sum of squares. Whereas In Ridge regression, the penalty is the sum of the squares of the coefficients.
- Ridge regression adds the “squared magnitude” of coefficient as a penalty term to the loss function.
- Lasso Regression adds the “absolute value of magnitude” of coefficient as a penalty term to the loss function.

As know that ridge regression can’t zero coefficients. Here, we either select all the coefficients or none of them whereas LASSO does both parameter shrinkage and variable selection automatically because it zeroes out the coefficients of collinear variables.

Role of Hyper Parameter in Regularization Task:

Hyperparameter is a parameter whose value is used to control the learning process.

When using the regularization method then have to need to decide how much weight we want to give to that regularization method. We can pick larger or smaller values for your complexity penalty depending on how much we think overfitting is going to be a problem for our current use case. This exposes one of the open secrets of machine learning its goal is to get the computer to learn how to make decisions automatically, but there are values impacting performance that must be chosen. Every machine learning algorithm has these values, called hyperparameters. These hyperparameters are values or functions that govern the way the algorithm behaves.

Question - 3

- when there are more and more dimensions, computations within that space become more and more expensive. This is when the kernel trick comes in. It allows us to operate in the original feature space without computing the coordinates of the data in a higher-dimensional space.
- increase C, but the value of *C* depends upon the *database.*

C is a regularization parameter that controls the trade-off between achieving a low training error and a low testing error that is the ability to generalize your classifier to unseen data.

- For a higher gamma, the model will capture the shape of the dataset well.

The **gamma** parameter defines how far the influence of a single training example reaches

Question - 4

C tells the algorithm how much care about misclassified points. Large values for the C parameter can drastically overfit the data and low values can underfit the data. Therefore, if the dataset is underfitting, we should increase the value of C gradually.

Gamma decides that how much curvature we want in a decision boundary. Gamma high means more curvature. Gamma low means less curvature. As the gamma decreases, the regions separating different classes get more generalized. Very large gamma values result in too specific class regions (overfitting). Therefore, if the data is underfitting, we should increase the gamma value.

Question - 5

The theoretical **maximum depth** a decision tree can achieve is one less than the number of training samples, but no algorithm will let you reach this point for obvious reasons, one big reason being overfitting. Note here that it is the number of training samples and not the number of features because the data can be split on the same feature multiple times.

In case of default None case, if don't specify a depth for the tree, sci-kit-learn will expand the nodes until all leaves are pure, meaning the leaf will only have labels if choose default for the min_samples_leaf, where the default value is one. Most of these hyperparameters are tied to one another. On the other hand, if specify a min_samples_split, which will look at next, the nodes will be expanded until all leaves contain less than the minimum number of samples. Scikit-learn will pick one over the other depending on which gives the maximum depth for the tree. There are a lot of moving parts here, min_samples_split and min_samples_leaf. If just take the max_depth in isolation and look for a change in the model, so after going through min_samples_split and min_samples_leaf can get a better intuition of how all these come together.

It is also bad to have a very low depth because the model will underfit so to find the best value, experiment because overfitting and underfitting are very subjective to a dataset, there is no one value fits all solution. So let the model decide the **max_depth** first, and then by comparing train and test scores, then look for overfitting or underfitting and depending on the degree decrease or increase the **max_depth**.

What mean by the impurity of a node in a decision tree:

The *node impurity* is a measure of the homogeneity of the labels at the node. The current implementation provides two impurity measures for classification (Gini impurity and entropy) and one impurity measure for regression (variance).

Impurity	**Task**	**Formula**	**Description**
Gini impurity	Classification	∑Ci=1fi(1−fi)∑i=1Cfi(1−fi)	fifi is the frequency of label ii at a node and CC is the number of unique labels.
Entropy	Classification	∑Ci=1−filog(fi)∑i=1C−filog(fi)	fifi is the frequency of label ii at a node and CC is the number of unique labels.
Variance	Regression	1N∑Ni=1(yi−μ)21N∑i=1N(yi−μ)2	yiyi is label for an instance, NN is the number of instances and μμ is the mean given by 1N∑Ni=1yi1N∑i=1Nyi.

The *information gain* is the difference between the parent node impurity and the weighted sum of the two-child node impurities. Assuming that a split ss partitions the dataset DD of size NN into two datasets DleftDleft and DrightDright of sizes NleftNleft and NrightNright, respectively, the information gain is:

IG(D,s)=Impurity(D)−NleftNImpurity(Dleft)−NrightNImpurity(Dright)

Question - 6

Random Forest Algorithm:

- Random Forest is a computationally efficient technique that can operate quickly over large datasets.
- Random forest is a supervised learning algorithm.
- The "forest" it builds, is an ensemble of decision trees, usually trained with the “bagging” method.
- The idea of the bagging method is a combination of learning models that increases the overall result.
- It can be used for both classification and regression problems.

Decision trees are types of models used for both classification and regression. And several decision trees make a **random forest**.

It is suggested that a random forest should have a number of trees between 64 - 128 trees.

The number of decision trees we need to use to get a good result:

- A threshold from which increasing the number of trees would bring no significant performance gain, and would only increase the computational cost.
- As the number of trees grows, it does not always mean the performance of the forest is significantly better than previous forests (fewer trees), and
- Doubling the number of trees is worthless.
- It is also possible to state there is a threshold beyond which there is no significant gain unless a huge computational environment is available.

Random Forests can also be used for regression tasks. A random forest's nonlinear nature can give it a leg up over linear algorithms, making it a great option.

Random Forest algorithm can be used for both classification and regression tasks. It will provide higher accuracy through cross-validation. Random forest classifier will handle the missing values and maintain the accuracy of a large proportion of data.

Random forest algorithm for regression problem:

- A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap, commonly known as bagging.
- The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.
- Random Forest has multiple decision trees as base learning models. We randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model.
- This part is called Bootstrap.

- In the case of a regression problem, the final output is the mean of all the outputs. This part is Aggregation

Creating and training the model:

```
from sklearn.ensemble import RandomForestRegressor

regressor = RandomForestRegressor(n_estimators=10, random_state=42)

regressor.fit(x,y)
```