Machine Learning Prep Guide

Anjali Chauhan

2022-04-27

Contents

Chapter 1

Prerequisites

```
install.packages("bookdown")
# or the development version
# devtools::install_github("rstudio/bookdown")
```

Chapter 2

ML Models

2.1 Linear Regression

Linear Regression involves finding a ‘line of best fit’ that represents a dataset using the least squares method. The least squares method involves finding a linear equation that minimizes the sum of squared residuals. A residual is equal to the actual minus predicted value.

To give an example, the red line is a better line of best fit than the green line because it is closer to the points, and thus, the residuals are smaller.

Linear Regression is one of the most fundamental algorithms used to model relationships between a dependent variable and one or more independent variables. In simpler terms, it involves finding the ‘line of best fit’ that represents two or more variables.

The line of best fit is found by minimizing the squared distances between the points and the line of best fit — this is known as minimizing the sum of squared residuals. A residual is simply equal to the predicted value minus the actual value.

2.2 What’s the difference between Linear Regression and Logistic Regression?

Linear Regression is used to predict a continuous variable and is mainly used to solve regression problems. Linear regression finds the best fit line by which the output numerical value can be predicted.

Logistic Regression is used to predict categorical values and is mainly used in classification problems. Logistic regression produces an S curve that classifies, the output is binary or categories.

2.3 What is overfitting?

Overfitting is an error where the model ‘fits’ the data too well, resulting in a model with high variance and low bias. As a consequence, an overfit model will inaccurately predict new data points even though it has a high accuracy on the training data.

Overfitting is a modeling error when a function fits the data too closely, resulting in high levels of error when new data is introduced to the model.

There are a number of ways that you can prevent overfitting of a model:

- **Cross-validation**: Cross-validation is a technique used to assess how well a model performs on a new independent dataset. The simplest example of cross-validation is when you split your data into two groups: training data and testing data, where you use the training data to build the model and the testing data to test the model.
- **Regularization**: Overfitting occurs when models have higher degree polynomials. Thus, regularization reduces overfitting by penalizing higher degree polynomials.
- **Reduce the number of features**: You can also reduce overfitting by simply reducing the number of input features. You can do this by manually removing features, or you can use a technique, called Principal Component Analysis, which projects higher dimensional data (eg. 3 dimensions) to a smaller space (eg. 2 dimensions).
- **Ensemble Learning Techniques**: Ensemble techniques take many weak learners and converts them into a strong learner through bagging and boosting. Through bagging and boosting, these techniques tend to overfit less than their alternative counterparts.

Overfitting is an error where the model ‘fits’ the data too well, resulting in a model with high variance and low bias. As a consequence, an overfit model will inaccurately predict new data points even though it has a high accuracy on the training data.

2.4 What is the bias-variance tradeoff?

The bias of an estimator is the difference between the expected value and true value. A model with a high bias tends to be oversimplified and results in underfitting. Variance represents the model’s sensitivity to the data and the noise. A model with high variance results in overfitting.

Therefore, the bias-variance tradeoff is a property of machine learning models in which lower variance results in higher bias and vice versa. Generally, an optimal balance of the two can be found in which error is minimized.

2.5 What are ridge and lasso regression and what are the differences between them?

Both L1 and L2 regularization are methods used to reduce the overfitting of training data. Least Squares minimizes the sum of the squared residuals, which can result in low bias but high variance.

L2 Regularization, also called ridge regression, minimizes the sum of the squared residuals plus lambda times the slope squared. This additional term is called the Ridge Regression Penalty. This increases the bias of the model, making the fit worse on the training data, but also decreases the variance.

If you take the ridge regression penalty and replace it with the absolute value of the slope, then you get Lasso regression or L1 regularization.

L2 is less robust but has a stable solution and always one solution. L1 is more robust but has an unstable solution and can possibly have multiple solutions.

2.6 What's the difference between L2 and L1 regularization?

- **Penalty terms**: L1 regularization uses the sum of the absolute values of the weights, while L2 regularization uses the sum of the weights squared.

- **Feature selection**: L1 performs feature selection by reducing the coefficients of some predictors to 0, while L2 does not. Computational efficiency: L2 has an analytical solution, while L1 does not.

- **Multicollinearity**: L2 addresses multicollinearity by constraining the coefficient norm.

- L1 effectively removes features that are unimportant, and doing this too aggressively can lead to underfitting. L2 weighs each feature instead of removing them entirely, which can lead to better accuracy. Briefly, L1 removes features while L2 doesn't, L2 regulates their weights instead.

2.7 What's Regularization? and what's the difference between L1 and L2 regularization?

Regularization in machine learning is the process of regularizing the parameters that constrain, regularizes, or shrinks the coefficient estimates towards zero. In other words, this technique discourages learning a more complex or flexible model, avoiding the risk of Overfitting. Regularization basically adds the penalty as model complexity increases which can help avoid overfitting.

2.7.1 Ridge Regression

Ridge regression, also known as L2 Regularization, is a regression technique that introduces a small amount of bias to reduce overfitting. It does this by minimizing the sum of squared residuals plus a penalty, where the penalty is equal to lambda times the slope squared. Lambda refers to the severity of the penalty.

Without a penalty, the line of best fit has a steeper slope, which means that it is more sensitive to small changes in X. By introducing a penalty, the line of best fit becomes less sensitive to small changes in X. This is the idea behind ridge regression.

2.7.2 Lasso Regression

Lasso Regression, also known as L1 Regularization, is similar to Ridge regression. The only difference is that the penalty is calculated with the absolute value of the slope instead.

2.8 Can we use L1 regularization for feature selection?

Yes, because the nature of L1 regularization will lead to sparse coefficients of features. Feature selection can be done by keeping only features with non-zero coefficients.

2.9 When do we need to perform feature normalization for linear models? When it's okay not to do it?

Feature normalization is necessary for L1 and L2 regularizations. The idea of both methods is to penalize all the features relatively equally. This can't be done effectively if every feature is scaled differently.

Linear regression without regularization techniques can be used without feature normalization. Also, regularization can help to make the analytical solution more stable, — it adds the regularization matrix to the feature matrix before inverting it.

2.10 Logistic Regression

Logistic Regression is a classification technique that also finds a 'line of best fit'. However, unlike linear regression where the line of best fit is found using least squares, logistic regression finds the line (logistic curve) of best fit using maximum likelihood. This is done because the y value can only be one or zero.

Logistic regression is similar to linear regression but is used to model the probability of a discrete number of outcomes, typically two. For example, you might want to predict whether a person is alive or dead given their age.

At a glance, logistic regression sounds much more complicated than linear regression, but really only has one extra step.

First, you calculate a score using an equation similar to the equation for the line of best fit for linear regression.

The extra step is feeding the score that you previously calculated in the sigmoid function below so that you get a probability in return. This probability can then be converted to a binary output, either 1 or 0.

To find the weights of the initial equation to calculate the score, methods like gradient descent or maximum likelihood are used. Since it's beyond the scope of this article, I won't go into much more detail, but now you know how it works!

2.11 What is logistic regression? Or State an example when you have used logistic regression recently.

Logistic Regression often referred as logit model is a technique to predict the binary outcome from a linear combination of predictor variables. For example, if you want to predict whether a particular political leader will win the election or not. In this case, the outcome of prediction is binary i.e. 0 or 1 (Win/Lose). The predictor variables here would be the amount of money spent for election campaigning of a particular candidate, the amount of time spent in campaigning, etc.

Chapter 3

Metrics

3.1 Accuracy:

(TP+TN/(TP+FP+FN+TN)

- **WHAT**: Accuracy is the proportion of true results among the total number of cases examined.
- **WHEN**: Accuracy is a valid choice of evaluation for classification problems which are well balanced and not skewed or No class imbalance.
- **AWARE**: What if we are predicting if an asteroid will hit the earth? Just say No all the time. And you will be 99% accurate. My model can be reasonably accurate, but not at all valuable

3.2 Precision:

(TP)/(TP+FP)

- **WHAT**: It answers the question: what proportion of predicted Positives is truly Positive?
  - Asteroid problem, we never predicted a true positive, precision = 0
- **WHEN**: Precision is a valid choice of evaluation metric when we want to be very sure of our prediction. For example: If we are building a system to predict if we should decrease the credit limit on a particular account, we want to be very sure about our prediction or it may result in customer dissatisfaction.

- **AWARE**: Being very precise means our model will leave a lot of credit defaulters untouched and hence lose money.

3.3 Recall:

(TP)/(TP+FN)

- **WHAT**: answers a different question: what proportion of actual Positives is correctly classified?
  - Asteroid problem, we never predicted a true positive, recall = 0
- **WHEN**: Recall is a valid choice of evaluation metric when we want to capture as many positives as possible. For example: If we are building a system to predict if a person has cancer or not, we want to capture the disease even if we are not very sure.
- **AWARE**: Recall is 1 if we predict 1 for all examples.

3.4 F1 Score:

2 **(precision*recall)/(precision + recall)***

- **WHAT**: Here we utilize the tradeoff of precision vs. recall. The F1 score is a number between 0 and 1 and is the harmonic mean of precision and recall.
  - The F1 score is a number between 0 and 1 and is the harmonic mean of precision and recall.
  - We are predicting if an asteroid will hit the earth or not.
  - So if we say "No" for the whole training set. Our precision here is 0. What is the recall of our positive class? It is zero. What is the accuracy? It is more than 99%.
  - And hence the F1 score is also 0. And thus we get to know that the classifier that has an accuracy of 99% is basically worthless for our case. And hence it solves our problem.
- **WHEN**: We want to have a model with both good precision and recall.
- F1 score sort of maintains a balance between the precision and recall for your classifier. If your precision is low, the F1 is low and if the recall is low again your F1 score is low.

- **EXAMPLE**: If you are a police inspector and you want to catch criminals, you want to be sure that the person you catch is a criminal (Precision) and you also want to capture as many criminals (Recall) as possible. The F1 score manages this tradeoff.

- **AWARE**: The main problem with the F1 score is that it gives equal weight to precision and recall. We might sometimes need to include domain knowledge in our evaluation where we want to have more recall or more precision.To solve this, we can do this by creating a weighted F1 metric as below where beta manages the tradeoff between precision and recall.

  – ROC stands for Receiver Operating Characteristics. The diagrammatic representation that shows the contrast between true positive rate vs false positive rate. It is used when we need to predict the probability of the binary outcome. AUC is area under ROC curve. It represents degree of separability. It is used to value how much model is capable of distinguishing between classes. Higher value the better (0,1)

3.5 Which metrics to use when (classification)?

Accuracy, Precision, and Recall:

3.5.1 Accuracy

Accuracy is the quintessential classification metric. It is pretty easy to understand. And easily suited for binary as well as a multiclass classification problem.

Accuracy = (TP+TN)/(TP+FP+FN+TN)

Accuracy is the proportion of true results among the total number of cases examined.

When to use?

Accuracy is a valid choice of evaluation for classification problems which are well balanced and not skewed or No class imbalance.

Caveats

Let us say that our target class is very sparse. Do we want accuracy as a metric of our model performance? What if we are predicting if an asteroid will hit the earth? Just say No all the time. And you will be 99% accurate. My model can be reasonably accurate, but not at all valuable.

3.5.2 Precision

Let's start with precision, which answers the following question: what proportion of predicted Positives is truly Positive?

Precision = (TP)/(TP+FP)

In the asteroid prediction problem, we never predicted a true positive.

And thus precision=0

When to use?

Precision is a valid choice of evaluation metric when we want to be very sure of our prediction. For example: If we are building a system to predict if we should decrease the credit limit on a particular account, we want to be very sure about our prediction or it may result in customer dissatisfaction.

Caveats

Being very precise means our model will leave a lot of credit defaulters untouched and hence lose money.

3.5.3 Recall

Another very useful measure is recall, which answers a different question: what proportion of actual Positives is correctly classified?

Recall = (TP)/(TP+FN)

In the asteroid prediction problem, we never predicted a true positive.

And thus recall is also equal to 0.

When to use?

Recall is a valid choice of evaluation metric when we want to capture as many positives as possible. For example: If we are building a system to predict if a person has cancer or not, we want to capture the disease even if we are not very sure.

Caveats

Recall is 1 if we predict 1 for all examples. And thus comes the idea of utilizing tradeoff of precision vs. recall — F1 Score.

3.5.4 F1 Score

This is my favorite evaluation metric and I tend to use this a lot in my classification projects. The F1 score is a number between 0 and 1 and is the harmonic mean of precision and recall.

Let us start with a binary prediction problem. We are predicting if an asteroid will hit the earth or not. So if we say “No” for the whole training set. Our precision here is 0. What is the recall of our positive class? It is zero. What is the accuracy? It is more than 99%. And hence the F1 score is also 0. And thus we get to know that the classifier that has an accuracy of 99% is basically worthless for our case. And hence it solves our problem.

When to use?

We want to have a model with both good precision and recall.

3.5.5 Precision-Recall Tradeoff

Simply stated the F1 score sort of maintains a balance between the precision and recall for your classifier. If your precision is low, the F1 is low and if the recall is low again your F1 score is low.

If you are a police inspector and you want to catch criminals, you want to be sure that the person you catch is a criminal (Precision) and you also want to capture as many criminals (Recall) as possible. The F1 score manages this tradeoff.

How to Use?

You can calculate the F1 score for binary prediction problems using:

```
# from sklearn.metrics import f1_score

# y_true = [0, 1, 1, 0, 1, 1]
# y_pred = [0, 0, 1, 0, 0, 1]
# f1_score(y_true, y_pred)
```

This is one of my functions which I use to get the best threshold for maximizing F1 score for binary predictions. The below function iterates through possible threshold values to find the one that gives the best F1 score.

```
## y_pred is an array of predictions
# def bestThresshold(y_true,y_pred):
#     best_thresh = None
#     best_score = 0
#     for thresh in np.arange(0.1, 0.501, 0.01):
#         score = f1_score(y_true, np.array(y_pred)>thresh)
#         if score > best_score:
#             best_thresh = thresh
#             best_score = score
#     return best_score , best_thresh
```

Caveats

The main problem with the F1 score is that it gives equal weight to precision and recall. We might sometimes need to include domain knowledge in our evaluation where we want to have more recall or more precision.

To solve this, we can do this by creating a weighted F1 metric as below where beta manages the tradeoff between precision and recall.

Here we give β times as much importance to recall as precision.

```
# from sklearn.metrics import fbeta_score

# y_true = [0, 1, 1, 0, 1, 1]
# y_pred = [0, 0, 1, 0, 0, 1]
# fbeta_score(y_true, y_pred,beta=0.5)
```

F1 Score can also be used for Multiclass problems.

3.5.6 Log Loss/Binary Crossentropy

Log loss is a pretty good evaluation metric for binary classifiers and it is sometimes the optimization objective as well in case of Logistic regression and Neural Networks. Binary Log loss for an example is given by the below formula where p is the probability of predicting 1.

As you can see the log loss decreases as we are fairly certain in our prediction of 1 and the true label is 1.

When to Use?

When the output of a classifier is prediction probabilities. Log Loss takes into account the uncertainty of your prediction based on how much it varies from the actual label. This gives us a more nuanced view of the performance of our model. In general, minimizing Log Loss gives greater accuracy for the classifier.

How to Use?

```
# from sklearn.metrics import log_loss
## where y_pred are probabilities and y_true are binary class labels
# log_loss(y_true, y_pred, eps=1e-15)
```

Caveats

It is susceptible in case of imbalanced datasets. You might have to introduce class weights to penalize minority errors more or you may use this after balancing your dataset.

3.5.7 Categorical Cross Entropy

The log loss also generalizes to the multiclass problem. The classifier in a multiclass setting must assign a probability to each class for all examples. If there are N samples belonging to M classes, then the Categorical Crossentropy is the summation of -ylogp values:

- y__ij is 1 if the sample i belongs to class j else 0
- p__ij is the probability our classifier predicts of sample i belonging to class j.

When to Use?

When the output of a classifier is multiclass prediction probabilities. We generally use Categorical Crossentropy in case of Neural Nets. In general, minimizing Categorical cross-entropy gives greater accuracy for the classifier.

How to Use?

```
# from sklearn.metrics import log_loss
## Where y_pred is a matrix of probabilities with shape = (n_samples, n_classes) and y_true is an
# log_loss(y_true, y_pred, eps=1e-15)
```

Caveats:

It is susceptible in case of imbalanced datasets.

3.5.8 AUC

AUC is the area under the ROC curve.

AUC ROC indicates how well the probabilities from the positive classes are separated from the negative classes

3.5.9 ROC curve

We have got the probabilities from our classifier. We can use various threshold values to plot our sensitivity(TPR) and (1-specificity)(FPR) on the cure and we will have a ROC curve; where True positive rate or TPR is just the proportion of trues we are capturing using our algorithm.

Sensitivity = TPR(True Positive Rate)

Recall = TP/(TP+FN)

and False positive rate or FPR is just the proportion of false we are capturing using our algorithm.

1- Specificity = FPR(False Positive Rate)= FP/(TN+FP)

Here we can use the ROC curves to decide on a Threshold value.

The choice of threshold value will also depend on how the classifier is intended to be used.

If it is a cancer classification application you don't want your threshold to be as big as 0.5. Even if a patient has a 0.3 probability of having cancer you would classify him to be 1.

Otherwise, in an application for reducing the limits on the credit card, you don't want your threshold to be as less as 0.5. You are here a little worried about the negative effect of decreasing limits on customer satisfaction.

When to Use?

AUC is scale-invariant. It measures how well predictions are ranked, rather than their absolute values. So, for example, if you as a marketer want to find a list of users who will respond to a marketing campaign. AUC is a good metric to use since the predictions ranked by probability is the order in which you will create a list of users to send the marketing campaign.

Another benefit of using AUC is that it is classification-threshold-invariant like log loss. It measures the quality of the model's predictions irrespective of what classification threshold is chosen, unlike F1 score or accuracy which depend on the choice of threshold.

How to Use?

```
# import numpy as np
# from sklearn.metrics import roc_auc_score
# y_true = np.array([0, 0, 1, 1])
# y_scores = np.array([0.1, 0.4, 0.35, 0.8])
# print(roc_auc_score(y_true, y_scores))
```

Caveats

Sometimes we will need well-calibrated probability outputs from our models and AUC doesn't help with that.

3.6 What is the difference between precision and recall?

Recall attempts to answer "What proportion of actual positives was identified correctly?"

Precision attempts to answer "What proportion of positive identifications was actually correct?"

3.7 Precision-recall trade-off

Tradeoff means increasing one parameter would lead to decreasing of other. Precision-recall tradeoff occur due to increasing one of the parameter(precision or recall) while keeping the model same.

In an ideal scenario where there is a perfectly separable data, both precision and recall can get maximum value of 1.0. But in most of the practical situations, there is noise in the dataset and the dataset is not perfectly separable. There might be some points of positive class closer to the negative class and vice versa. In such cases, shifting the decision boundary can either increase the precision or recall but not both. Increasing one parameter leads to decreasing of the other.

3.8 What is the ROC curve? When to use it?

ROC stands for Receiver Operating Characteristics. The diagrammatic representation that shows the contrast between true positive rate vs false positive rate. It is used when we need to predict the probability of the binary outcome.

3.9 What is AUC (AU ROC)? When to use it?

AUC stands for Area Under the ROC Curve. ROC is a probability curve and AUC represents degree or measure of separability. It's used when we need to value how much model is capable of distinguishing between classes. The value is between 0 and 1, the higher the better.

3.10 We have two models, one with 85% accuracy, one 82%. Which one do you pick?

If we only care about the accuracy of the model then we would choose the one with 85%. But if an interviewer were to ask this, it would probably be a good idea to get more context, i.e. what the model is trying to predict. This will give us a better idea whether the evaluation metric should indeed be accuracy or another metric like recall or f1 score.

3.11 Is accuracy always a good metric?

Accuracy is not a good performance metric when there is imbalance in the dataset. For example, in binary classification with 95% of A class and 5% of B class, a constant prediction of A class would have an accuracy of 95%. In case of imbalance dataset, we need to choose Precision, recall, or F1 Score depending on the problem we are trying to solve.

3.12 What is a confusion matrix?

A confusion matrix, also known as an error matrix, is a summarized table used to assess the performance of a classification model. The number of correct and incorrect predictions are summarized with count values and broken down by each class.

- **True positive(TP)** — Correct positive prediction
- **False positive(FP)** — Incorrect positive prediction
- **True negative(TN)** — Correct negative prediction
- **False negative(FN)** — Incorrect negative prediction

3.13 How to check if the regression model fits the data well?

There are a couple of metrics that you can use:

- **R-squared/Adjusted R-squared**: Relative measure of fit. This was explained in a previous answer
- **F1 Score**: Evaluates the null hypothesis that all regression coefficients are equal to zero vs the alternative hypothesis that at least one doesn't equal zero
- **RMSE**: Absolute measure of fit.

3.14 Model Evaluation

To evaluate a regression model, you can calculate its R-squared, which tells us how much of the variability in the data that the model accounts for. For example, if a model has an R-squared of 80%, then 80% of the variation in the data can be explained by the model.

The adjusted R-squared is a modified version of r-squared that adjusts for the number of predictors in the model; it increases if the new term improves the model more than would be expected by chance and vice versa.

Chapter 4

Clustering

We describe our methods in this chapter.

Math can be added in body using usual syntax like this

4.1 math example

p is unknown but expected to be around 1/3. Standard error will be approximated

$$SE = \sqrt(\frac{p(1-p)}{n}) \approx \sqrt{\frac{1/3(1-1/3)}{300}} = 0.027$$

You can also use math in footnotes like this[1].

We will approximate standard error to 0.027[2]

[1]where we mention $p = \frac{a}{b}$

[2]p is unknown but expected to be around 1/3. Standard error will be approximated

$$SE = \sqrt(\frac{p(1-p)}{n}) \approx \sqrt{\frac{1/3(1-1/3)}{300}} = 0.027$$

Chapter 5

Data Preprocessing

5.1 What do you mean by Feature Splitting?

A feature splitting is an approach to generate some new features from the existing one to improve the performance of our model.

For Example, splitting names of objects into two parts- first and last names.

5.2 What are the feature selection methods used to select the right variables?

There are two types of methods for feature selection: filter methods and wrapper methods.

Filter methods include the following:

- Linear discrimination analysis
- ANOVA
- Chi-Square

Wrapper methods include the following:

- Forward Selection: We test one feature at a time and keep adding them until we get a good fit
- Backward Selection: We test all the features and start removing them to see what works better

5.3 How do you select the important features while working on a dataset?

There are various methods to select important features from a data set that include the following:

- **Multicollinearity**: Identify and discard correlated variables before finalizing important variables.
- **Using the Linear Regression model**: More explanatory variables could be selected based on the p-values obtained from the Linear Regression.
- **Wrapper Methods**: Forward, Backward, and Stepwise selection
- **Regularization**: Lasso Regression
- **Ensemble Technique**: Apply Random Forest and then plot the variable chart
- **Information Gain**: Top features can be selected based on information gain for the available set of features.

5.4 What are some of the steps for data wrangling and data cleaning before applying machine learning algorithms?

There are many steps that can be taken when data wrangling and data cleaning. Some of the most common steps are listed below:

- **Data profiling**: Almost everyone starts off by getting an understanding of their dataset. More specifically, you can look at the shape of the dataset with .shape and a description of your numerical variables with .describe().
- **Data visualizations**: Sometimes, it's useful to visualize your data with histograms, boxplots, and scatterplots to better understand the relationships between variables and also to identify potential outliers.
- **Syntax error**: This includes making sure there's no white space, making sure letter casing is consistent, and checking for typos. You can check for typos by using .unique() or by using bar graphs.
- **Standardization or normalization**: Depending on the dataset your working with and the machine learning method you decide to use, it may be useful to standardize or normalize your data so that different scales of different variables don't negatively impact the performance of your model.

- **Handling null values**: There are a number of ways to handle null values including deleting rows with null values altogether, replacing null values with the mean/median/mode, replacing null values with a new category (eg. unknown), predicting the values, or using machine learning models that can deal with null values. Read more here.
- **Other things include**: removing irrelevant data, removing duplicates, and type conversion.

5.5 What are the missing values? How do you handle missing values?

In the data cleaning process, we can see there are lots of missing values that are not filled or collected during the survey for data collection. So, we have to handle such missing values using the following methods:

- **Delete the missing records**: Ignoring such rows or dropping such records.
- **Central Tendency**: Fill values with mean, mode, and median.
- **Different mean for different classes**: Fill values using mean but for different classes, different means can be used.
- **Model Building**: You can also fill the most probable value using regression, Bayesian formula, or decision tree, KNN, and Prebuilt imputing libraries.
- **Use constant Value**: Fill with a constant value.
- To fill the values manually.

5.6 Name two useful methods of pandas that are useful to handle the missing values.

The two useful methods of Pandas Library are –

- isnull( )
- dropna( )

These help to find the columns of data with missing or corrupted data and drop those values. In cases where we want to fill the invalid values with a placeholder value such as 0, we can use the fillna() method.

5.7 Give several ways to deal with missing values

There are a number of ways to handle null values including the following:

- You can omit rows with null values altogether
- You can replace null values with measures of central tendency (mean, median, mode) or replace it with a new category (eg. 'None')
- You can predict the null values based on other variables. For example, if a row has a null value for weight, but it has a value for height, you can replace the null value with the average weight for that given height.
- Lastly, you can leave the null values if you are using a machine learning model that automatically deals with null values.

5.8 Explain One-hot encoding and Label Encoding. Does the dimensionality of the dataset increase or decrease after applying these techniques?

Both One-hot encoding and Label Encoding are used to encode the categorical variables to numerical variables so that we can feed that encoded data to our ML algorithms and be able to find insights about the data.

One-hot encoding is the representation of categorical variables as binary vectors while Label Encoding converts labels or words into numeric form.

One-hot encoding increases the dimensionality of the data set while Label encoding doesn't affect the dimensionality of the data set since One-hot encoding creates a new variable for each level in the variable whereas, in Label encoding, the levels of a variable get encoded as 1 and 0 in replacement of the same column.

5.9 Why do we need one-hot encoding?

If we simply encode categorical variables with a Label encoder, they become ordinal which can lead to undesirable consequences. In this case, linear models will treat category with id 4 as twice better than a category with id 2. One-hot encoding allows us to represent a categorical variable in a numerical vector space which ensures that vectors of each category have equal distances between each other. This approach is not suited for all situations, because by using it with categorical variables of high cardinality (e.g. customer id) we will encounter problems that come into play because of the curse of dimensionality.

5.10 How can you determine which features are the most important in your model?

In applied machine learning, success depends significantly on the quality of data representation (features). Highly correlated features can make learning/sorting steps in the classification module easy. Conversely if label classes are a very complex function of the features, it can be impossible to build a good model [Dom 2012]. Thus a so-called feature engineering, a process of transforming data into features that are most relevant to the problem, is often needed.

A feature selection scheme often involves techniques to automatically select salient features from a large exploratory feature pool. Redundant and irrelevant features are well known to cause poor accuracy so discarding these features should be the first task. The relevance is often scored using mutual information calculation. Furthermore, input features should thus offer a high level of discrimination between classes. The separability of features can be measured by distance or variance ratio between classes. One recent work [Pham 2016] proposed a systematic voting based feature selection that is a data-driven approach incorporating above criteria. This can be used as a common framework for a wide class of problems.

Another approach is penalizing on the features that are not very important (e.g., yield a high error metric) when using regularization methods like Lasso or Ridge.

5.11 How can you choose a classifier based on training set size?

If training set is small, high bias/low variance models (e.g. Naïve Bayes) tend to perform better because they are less likely to be overfit.

If training set is large, low bias/high variance models (e.g. Logistic Regression) tend to perform better because they can reflect more complex relationships.

5.12 Is it good to perform scaling before the split or after the split by keeping train and test split criteria in mind?

Ideally, we need to do a scaling post-train and test split. Scaling post or pre-split should not make much difference when our data is closely packed.

5.13 When can a categorical value be treated as a continuous variable and what effect does it have when done so?

A categorical independent variable can be treated as a continuous one when the nature of data points the categorical data represents is ordinal in nature.

So, if the independent variable is having ordinal data then it can be treated as continuous and its inclusion in the model increases the performance of the model.

5.14 If we have a date column in our dataset, then how will you perform Feature Engineering?

From a date column, we can get lots of important features such as day of the week, day of the month, day of the quarter, and day of the year.

Moreover, we can extract the date, month, and year from that column also. All these features can impact your prediction and make our model robust.

For Example, Sales of the business can be impacted by month or day of the week.

5.15 How would you handle an imbalanced dataset?

An imbalanced dataset is when you have, For Example, a classification problem statement and let's 90% of the data is in one class. As a result, an accuracy of 90% can be skewed if you have no predictive power on the other category of data!

Here are a few suggestions to get rid of these problems:

- To collect more dataset that helps to even the imbalances in the dataset.
- Resample the dataset to correct for imbalances.
- Try a different algorithm altogether on your dataset.

So, it's important that you have a keen sense of what damage an unbalanced dataset can cause, and how to balance that.

5.16 How should you deal with unbalanced binary classification?

There are a number of ways to handle unbalanced binary classification (assuming that you want to identify the minority class):

- First, you want to reconsider the metrics that you'd use to evaluate your model. The accuracy of your model might not be the best metric to look at because and I'll use an example to explain why. Let's say 99 bank withdrawals were not fraudulent and 1 withdrawal was. If your model simply classified every instance as "not fraudulent", it would have an accuracy of 99%! Therefore, you may want to consider using metrics like precision and recall.
- Another method to improve unbalanced binary classification is by increasing the cost of misclassifying the minority class. By increasing the penalty of such, the model should classify the minority class more accurately.
- Lastly, you can improve the balance of classes by oversampling the minority class or by undersampling the majority class.

5.17 Which technique would you use in cases where the number of variables is greater than the number of observations in the dataset. Explain?

In cases where the number of variables is greater than the number of observations, it represents a high-dimensional dataset.

So, it is not possible to calculate a unique least-square coefficient estimate, and therefore we used the penalized regression methods like Least Angle regression(LARS), LASSO, or Ridge which seems to work well under these circumstances as they tend to shrink the coefficients in order to reduce the variance.

Moreover, in situations of higher variance of least square estimates Ridge Regression works better.

When the number of variables is greater than the number of observations, it represents a high dimensional dataset. In such cases, it is not possible to calculate a unique least square coefficient estimate. Penalized regression methods like LARS, Lasso or Ridge seem work well under these circumstances as they tend to shrink the coefficients to reduce variance. Whenever the least square estimates have higher variance, Ridge regression technique seems to work best.

5.18 What are 3 data preprocessing techniques to handle outliers?

- Winsorize (cap at threshold)
- Transform to reduce skew (using Box-Cox or similar)
- Remove outliers if you're certain they are anomalies or measurement errors.

##If a weight for one variable is higher than for another, can we say that this variable is more important?

Yes - if your predictor variables are normalized. Without normalization, the weight represents the change in the output per unit change in the predictor. If you have a predictor with a huge range and scale that is used to predict an output with a very small range - for example, using each nation's GDP to predict maternal mortality rates - your coefficient should be very small. That does not necessarily mean that this predictor variable is not important compared to the others.

5.19 How do we handle categorical variables in decision trees?

Some decision tree algorithms can handle categorical variables out of the box, others cannot. However, we can transform categorical variables, e.g. with a binary or a one-hot encoder.

##What happens when we have correlated features in our data?

In random forest, since random forest samples some features to build each tree, the information contained in correlated features is twice as much likely to be picked than any other information contained in other features.

In general, when you are adding correlated features, it means that they linearly contains the same information and thus it will reduce the robustness of your model. Each time you train your model, your model might pick one feature or the other to "do the same job" i.e. explain some variance, reduce entropy, etc.

5.20 For a classification problem, how will you know which Machine Learning Algorithm to Choose?

There is no fixed rule of thumb to choose an algorithm for a classification problem, but the following points can be kept in mind while selecting an algorithm:

- If accuracy is our major focus: Test different algorithms and cross-validate them.
- What If we have a small training dataset: Use models that have low variance and high bias.
- If we have a large training dataset: Use models that have high variance and little bias.

5.21 How much data will you allocate for your training, validation and test sets?

There is no to the point answer to this question but there needs to be a balance/equilibrium when allocating data for training, validation and test sets.

If you make the training set too small, then the actual model parameters might have high variance. Also, if the test set is too small, there are chances of unreliable estimation of model performance. A general thumb rule to follow is to use 80: 20 train/test spilt. After this the training set can be further split into validation sets.

5.22 In unsupervised learning, if a ground truth about a dataset is unknown, how can we determine the most useful number of clusters to be?

With supervised learning, the number of classes in a particular set of data is known outright, since each data instance in labeled as a member of a particular existent class. In the worst case, we can scan the class attribute and count up the number of unique entries which exist.

With unsupervised learning, the idea of class attributes and explicit class membership does not exist; in fact, one of the dominant forms of unsupervised learning – data clustering – aims to approximate class membership by minimizing interclass instance similarity and maximizing intraclass similarity. A major drawback with clustering can be the requirement to provide the number of classes which exist in the unlabeled dataset at the onset, in some form or another. If we are lucky, we may know the data's ground truth – the actual number of classes – beforehand. However, this is not always the case, for numerous reasons, one of which being that there may actually be no defined number of classes (and hence, clusters) in the data, with the whole point of the unsupervised learning task being to survey the data and attempt to impose some meaningful structure of optimal cluster and class numbers upon it.

5.23 Without knowing the ground truth of a dataset, then, how do we know what the optimal number of data clusters are?

As one may expect, there are actually numerous methods to go about answering this question. We will have a look at 2 particular popular methods for attempting to answer this question: the elbow method and the silhouette method.

5.23.1 The Elbow Method

The elbow method is often the best place to state, and is especially useful due to its ease of explanation and verification via visualization. The elbow method is interested in explaining variance as a function of cluster numbers (the k in k-means). By plotting the percentage of variance explained against k, the first N clusters should add significant information, explaining variance; yet, some eventual value of k will result in a much less significant gain in information, and it is at this point that the graph will provide a noticeable angle. This angle will be the optimal number of clusters, from the perspective of the elbow method.

It should be self-evident that, in order to plot this variance against varying numbers of clusters, varying numbers of clusters must be tested. Successive complete iterations of the clustering method must be undertaken, after which the results can be plotted and compared.

5.23.2 The Silhouette Method

The silhouette method measures the similarity of an object to its own cluster – called cohesion – when compared to other clusters – called separation. The silhouette value is the means for this comparison, which is a value of the range [-1, 1]; a value close to 1 indicates a close relationship with objects in its own cluster, while a value close to -1 indicates the opposite. A clustered set of data in a model producing mostly high silhouette values is likely an acceptable and appropriate model.

5.24 When should you use classification over regression?

Both classification and regression belong to the category of supervised machine learning algorithms. However, when the target is categorical, classification is used whereas when your target variable is continuous, regression is used. The classification produces discrete values and datasets to strict the categories, while regression gives continuous results that allow to better distinguish differences between individual points.

In order to represent the belonging of the data points to certain categories, we use classification over regression. For example, If you wanted to know whether a name was male or female rather than just how correlated they were with male and female names.

5.25 How can we use an unlabelled dataset(without having a target column) in Supervised Learning Algorithms?

To use a dataset without having an output column, we first give the input dataset into a clustering algorithm, which generates an optimal number of clusters, and then labels the cluster numbers as the new target variable.

Now, the dataset has both independent and dependent variables i.e, target column present.

So, this completes our objective to apply the supervised learning algorithms to the unlabeled data.

5.26 Is it possible to test the probability of improving the model accuracy without using cross-validation? If yes, please explain.

Yes, we can test for the probability of improving the accuracy of the model without using cross-validation techniques.

For doing this, We have to run our ML model for say K number of iterations, and then recording the accuracy. After that try to plot all the accuracies and remove the 5% of low probability values. Then, measure the left [low]and right [high] threshold, decided by the problem statement itself. Now, with the remaining 95% confidence, we can say that the model can go as low or as high i.e, determine the range.

5.27 What cross-validation technique would you use on a time series data set.

Instead of using k-fold cross-validation, you should be aware to the fact that a time series is not randomly distributed data — It is inherently ordered by chronological order.

In case of time series data, you should use techniques like forward chaining — Where you will be model on past data then look at forward-facing data.

fold 1: training[1], test[2]

fold 1: training[1 2], test[3]

fold 1: training[1 2 3], test[4]

fold 1: training[1 2 3 4], test[5]

5.28 How can you create a model with a very unbalanced dataset? For example, working with credit card fraud data and there are very few real fraud cases while the majority of the cases are non-fraudulent.

Creating a model with an unbalanced dataset will yield bad results in terms of favoring the more training data, in our case the non-fraudulent transactions. You should never create a model with an unbalanced dataset. The answer should be around trying to gather more balanced data and if not possible then oversample your data using SMOTE (Synthetic Minority Over Sampling) or Random Over Sampling (ROS).

5.28.1 SMOTE Techniques to Balance Datasets

The SMOTE technique creates new observations of the underrepresented class, in this case, the fraudulent observations. These synthetic observations are almost identical to the original fraudulent observations. This technique is expeditious, but the types of synthetic observations it produces are not as useful as the unique observations created by other oversampling techniques.

Chapter 6

Neural Networks

6.1 Briefly explain how a basic neural network works

At its core, a Neural Network is essentially a network of mathematical equations. It takes one or more input variables, and by going through a network of equations, results in one or more output variables.

In a neural network, there's an input layer, one or more hidden layers, and an output layer. The input layer consists of one or more feature variables (or input variables or independent variables) denoted as x1, x2, ..., xn. The hidden layer consists of one or more hidden nodes or hidden units. A node is simply one of the circles in the diagram above. Similarly, the output variable consists of one or more output units.

Like I said at the beginning, a neural network is nothing more than a network of equations. Each node in a neural network is composed of two functions, a linear function and an activation function. This is where things can get a little confusing, but for now, think of the linear function as some line of best fit. Also, think of the activation function like a light switch, which results in a number between 1 or 0.

6.2 Activation Functions

An activation function is like a light switch — it determines whether a neuron should be activated or not.

There are several types of activation functions, but the most popular activation function is the Rectified Linear Unit function, also known as the ReLU function.

6.2.1 What is the role of the activation function?

The purpose of the activation function is to introduce non-linearity into the output of a neuron. The activation function decides whether a neuron should be activated or not by calculating weighted sum and further adding bias with it.

6.2.2 Why Tanh activation function preferred over sigmoid?

Tanh function is called a shifted version of the sigmoid function. The output of Tanh centers around 0 and sigmoid's around 0.5. Tanh Convergence is usually faster if the average of each input variable over the training set is close to zero.

When you struggle to quickly find the local or global minimum, in such case Tanh can be helpful in faster convergence. The derivatives of Tanh are larger than Sigmoid that causes faster optimization of the cost function. Tanh and Sigmoid both suffered from vanishing gradient problems.

6.2.3 Why is the ReLU activation function is better than the sigmoid activation function?

Sigmoid function bounded between 0 and 1. It is differentiable, non-linear, and produces non-binary activations. But the problem with Sigmoid is the vanishing gradients.

ReLu(Rectified Linear Unit) is like a linearity switch. If you don't need it, you "switch" it off. If you need it, you "switch" it on. ReLu avoids the problem of vanishing gradient.

ReLu also provides the benefit of sparsity and sigmoids result in dense representations. Sparse representations are more useful than dense representations.

6.2.4 What is the use of the leaky ReLU function?

The main problem with ReLU is, it is not differentiable at 0 and may result in exploding gradients. To resolve this problem Leaky ReLu was introduced that is differentiable at 0. It provides small negative values when input is less than 0.

6.3 What is the “dying ReLU” problem in neural networks?

When ReLu provides output zero for any input(large negative biases). This problem will occur due to a high learning rate and large negative bias. Leaky ReLU is a commonly used method to overcome a dying ReLU problem. It adds a small negative slope to prevent the dying ReLU problem.

6.4 Why is Rectified Linear Unit a good activation function?

The Rectified Linear Unit, also known as the ReLU function, is known to be a better activation function than the sigmoid function and the tanh function because it performs gradient descent faster. Notice in the image to the left that when x (or z) is very large, the slope is very small, which slows gradient descent significantly. This, however, is not the case for the ReLU function.

6.5 What is the activation function? Why do we need them?

Activation functions are mathematical functions that transform the output of a neural network on a certain scale. It means it normalizes the output between range 0 and 1 or -1 and 1. Activation functions in neural networks introduce non-linearity. It helps neural networks to handle non-linear relationships.

6.6 What is backward and forward propagation?

Backpropagation traverses in the reverse direction. It computes the gradient(or delta rule) of parameters(weights and biases) in order to map the output layer to the input layer. The main objective of backpropagation is to minimize the error. This process will repeat until the error is minimized and final parameters will be used for producing the output.

Forward propagation or forward pass computes the intermediate values in order to map the input and output layer.

6.7 What is backpropagation in neural networks in a pure mathematical sense?

So backpropagation in Computer science is the algorithmic way in which we send the result of some computation back to the parent recursively.

In Machine learning, backpropagation sends feedback to the neural net.

The complete algorithm for this is known as Gradient Descent. The part of Gradient Descent(or similar algorithms) where you infer the error(usually with calculus) and correct it is known as Backpropagation.

So any training step includes calculating the gradient(differentiation in calculus) and then doing backpropagation(integrating the gradient to get back the way the weights should change).

6.8 Cost Function

A cost function for a neural network is similar to a cost function that you would use for any other machine learning model. It's a measure of how 'good" a neural network is in regards to the values that it predicts compared to the actual values. The cost function is inversely proportional to the quality of a model — the better the model, the lower the cost function and vice versa.

The purpose of a cost function is so that you have value to optimize. By minimizing the cost function of a neural network, you'll achieve the optimal weights and parameters of the model, thereby maximizing the performance of it. There are several commonly used cost functions, including the quadratic cost, cross-entropy cost, exponential cost, Hellinger distance, Kullback-Leibler divergence, etc.

The main objective of the neural network is to find the optimal set of weights and biases by minimizing the cost function. Cost function or loss function is a measure used to measure the performance of the neural network on test data set. It measures the ability to estimate the relationship between X and y. An example of cost functions is the mean square error.

6.9 Backpropagation

Backpropagation is an algorithm that closely ties with the cost function. Specifically, it is an algorithm that is used to compute the gradient of the cost function. It has adopted a lot of popularity and use due to its speed & efficiency compared to other approaches.

Its name stems from the fact that the calculation of the gradient starts with the gradient of the final layer of weights and moves backwards to the gradient of the first layer of weights. Consequently, the error at layer k is dependent on the next layer k+1.

Generally, backpropagation works as follows:

- Calculates the forward phase for each input-output pair
- Calculates the backward phase for each pair
- Combine the individual gradients
- Update the weights based on the learning rate and the total gradient

6.10 Difference between convex and non-convex cost function; what does it mean when a cost function is non-convex?

A convex function is one where a line drawn between any two points on the graph lies on or above the graph. It has one minimum.

A non-convex function is one where a line drawn between any two points on the graph may intersect other points on the graph. It characterized as "wavy".

When a cost function is non-convex, it means that there's a likelihood that the function may find local minima instead of the global minimum, which is typically undesired in machine learning models from an optimization perspective.

6.11 Convolutional Neural Networks

6.11.1 What is CNN? How does CNN work?

CNN is Feedforward neural network. CNN filters the raw image detail patterns and classifies them using a traditional neural network. Convolution focuses on small patches in the image and represents a weighted sum of image pixel values. It offers applications in Image recognition and object detection. It works in the following steps:

- Convolution Operation
- ReLu layer
- Pooling- Max and Min Pool
- Flattening

- Full connection

A Convolutional Neural Network (CNN) is a type of neural network that takes an input (usually an image), assigns importance to different features of the image, and outputs a prediction. What makes CNNs better than feedforward neural networks is that it better captures the spatial (pixel) dependencies throughout the image, meaning it can understand the composition of an image better.

For those who are interested, CNNs use a mathematical operation called convolution. Wikipedia defines convolution as a mathematical operation on two functions that produces a third function expressing how the shape of one is modified by the other. Thus, CNN's use convolution instead of general matrix multiplication in at least one of their layers.

TLDR: CNNs are a type of neural network that is mainly used for image classification.

6.11.2 What are Convolution layers?

The convolution layer is inspired by the visual cortex. It converts the image into layers, transforms into small images, and extracts features from images. It will sum up the results into a single output pixel. It captures the relationship between pixels and detects edge, blur, and sharpen features.

6.12 Recurrent Neural Networks

A Recurrent Neural Network (RNNs) is another type of neural network that works exceptionally well with sequential data due to its ability to ingest inputs of varying sizes. RNNs consider both the current input as well as previous inputs it was given, which means that the same input can technically produce a different output based on the previous inputs given.

Technically speaking, RNNs are a type of neural network where connections between the nodes form a digraph along a temporal sequence, allowing them to use their internal memory to process variable-length sequences of inputs.

TLDR: RNNs are a type of neural network that is mainly used for sequential or time-series data.

Recurrent neural networks, also known as RNNs, are a class of neural networks that allow previous outputs to be used as inputs while having hidden states.

They are commonly used to recognize the pattern of sequences in data, including time-series data, stock market data, etc.

6.13 Long Short-Term Memory Networks

Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Networks that addresses one of the shortfalls of regular RNNs: RNNs have short-term memory. Specifically, if a sequence is too long, i.e. if there is a lag greater than 5–10 steps, RNNs tend to dismiss information that was provided in the earlier steps. For example, if we fed a paragraph into an RNN, it may overlook information provided at the beginning of the paragraph.

Thus LSTMs were created to resolve this issue.

6.14 Weight Initialization

The point of weight initialization is to make sure that a neural network doesn't converge to a trivial solution. If the weights are all initialized to the same value(eg. equal to zero) then each unit will get exactly the same signal and every layer would behave as if it were a single cell.

Therefore, you want to randomly initialize the weights near zero, but not equal to zero. This is an expectation of the stochastic optimization algorithm that's used to train the model.

6.14.1 How are weights initialized in a Network?

The weights of a neural network MUST be initialized randomly because this is an expectation of stochastic gradient descent.

If you initialized all weights to the same value (i.e. zero or one), then each hidden unit will get exactly the same signal. For example, if all weights are initialized to 0, all hidden units will get zero signal.

6.15 Batch vs. Stochastic Gradient Descent

Batch gradient descent and stochastic gradient descent are two different methods used to compute the gradient.

Batch gradient descent simply computes the gradient using the whole dataset. It is much slower especially with larger datasets but is better for convex or smooth error manifolds. With stochastic gradient descent, the gradient is computed using a single training sample at a time. Because of this, it is computationally faster and less expensive. Consequently, however, when a global optimum is reached, it tends to bounce around — this results in a good solution but not an optimal solution.

6.16 Hyper-parameters

Hyper-parameters are the variables that regulate the network structure and the variables which govern how the network is trained. Common hyper-parameters include the following: - Model architecture parameters such as the number of layers, number of hidden units, etc.

- The learning rate (alpha)
- Network weight initialization
- Number of epochs (defined as one cycle through the whole training dataset)
- Batch size
- and more.

6.17 Learning Rate

The learning rate is a hyper-parameter used in neural networks that control how much to adjust the model in response to the estimated error each time the model weights are updated.

If the learning rate is too low, your model will train very slowly as minimal updates are made to the weights through each iteration. Thus, it would take many updates before reaching the minimum point.

If the learning rate is set too high, this causes undesirable divergent behavior to the loss function due to drastic updates in weights, and it may fail to converge.

6.18 What is Deep Learning?

Deep Learning is a subdomain of Machine Learning. In deep learning, a large number of layers in the architecture. These successive layers learn more complex patterns in the data. Deep Learning offers various applications in text, voice, image, and video data.

6.19 What is Gradient Descent? How does it work?

It is a first-order iterative optimization technique for finding the minimum of a function. It is an efficient optimization technique to find a local or global minimum. In gradient descent, gradient and step are taken at each point. It takes the current value of parameters and updates it with the help of gradient and step width. the gradient is recomputed again and steps decremented in each iteration. This process continues until the convergence achieved.

Types of Gradient Descent

- Full batch gradient descent uses a full dataset.
- Stochastic gradient descent uses a sample of the dataset.

6.19.1 Stochastic Gradient Descent

Stochastic Gradient Descent builds on top of Gradient Descent where it can work with complicated Cost Functions.

6.19.2 Gradient Descent Stuck in Local Minima and Misses True Minima.

The Gradient Descent works well only in case of convex cost functions with one minimum only. However, in the case of complicated Cost Functions, the Gradient Descent can easily get stuck in local minima which ruins your Neural Network learning.

6.19.3 Stochastic vs. Gradient Descent

To understand how Stochastic is different from Gradient Descent let's take an example. We will assume you have labeled data as rows and you're inputting them into your Neural Network for training.

6.20 What is the difference between model parameters and hyperparameters?

Model parameters are internal and can be estimated from data. Model hyperparameters are external to the model and can not be estimated from data.

6.21 What do you mean by Dropout and Batch Normalization?

Dropout is a technique for normalization. It drops out or deactivates some neurons from the neural network to remove the problem of overfitting. In other words, it introduces the noise in the neural network so that model is capable to generalize the model.

Normalization is used to reduce the algorithm time that spends on the oscillation on different values. It brings all the features on the same input scale.

Batch Normalization is also normalizing the values but at hidden states on small batches of data. The research has shown that removing Dropout with Batch Normalization improves the learning rate without loss in generalization.

6.22 What is vanishing gradient descent?

RNN follows the chain rule in its backpropagation. When one of the gradients approaches zero then all the gradient will move towards zero. This small value is not sufficient for training the model. Here, a small gradient means that weights and biases of the neural network will not be updated effectively.

Also, at hidden layers activation functions such as sigmoid function and Tanh causes small derivatives that decrease the gradient.

The solution to Vanishing Gradient Descent

- Use a different activation function such as ReLu(Rectified Linear Unit).
- Batch normalization can also solve this problem by simply normalizing the input space.
- Weight initialization
- LSTM

6.23 What is exploding gradient descent?

Exploding gradient is just an opposite situation of vanishing gradient. A too-large value of RNN causes powerful training. We can overcome this problem by using Truncated Backpropagation, penalties, Gradient Clipping.

Gradient is the direction and magnitude calculated during training of a neural network that is used to update the network weights in the right direction and by the right amount.

"Exploding gradients are a problem where large error gradients accumulate and result in very large updates to neural network model weights during training." At an extreme, the values of weights can become so large as to overflow and result in NaN values.

This has the effect of your model being unstable and unable to learn from your training data.

6.24 What is LSTM and BiLSTM?

LSTM is a special type of RNN. It also uses a chain-like structure but it has the capability to learn and remember long-term sequences. LSTM handles the issue of vanishing gradient. It keeps gradient step enough and therefore the short training and the high accuracy. It uses gated cells to write, read, erase the value. It has three gates: input, forget, and output gate.

BiLSTM learns sequential long terms in both directions. It captures the information from both the previous and next states. Finally, it merges the results of two states and produces output. It memorizes information about a sentence from both directions.

6.25 Explain gates used in LSTM with their functions.

LSTM has three gates: input, forget, and output gate. The input gate is used to add the information to the network, forget used to discard the information, and the output gate decides which information pass to the hidden and output layer.

6.26 What is the difference between LSTM and GRU?

GRU is also a type of RNN. It is slightly different from LSTM. The main difference between LSTM and GRU Gates is the number of gates.

- LSTM uses three gates(input, forget, and output gate) while GRU uses two gates(reset and update).
- In GRU, the update gate is similar to the input and forget gate of LSTM and the reset gate is another gate used to decide how much past information to forget.
- GRU is faster compared to LSTM.
- GRU needs fewer data to generalize.

6.27 What is padding?

Sometimes filter unable to fit the input image perfectly. We have two strategies for padding: Zero padding and valid padding. Zero paddings add zero so that the image filter fits the image. Valid padding drops the part of the image. (Drop the part of the image)

6.28 What are pooling and Flattening?

Pooling is used to reduce the spatial size and selects the important pixel values as features. It is also known as Downsampling. It also makes faster computation by reducing its dimension. Pooling summarizes the sub-region and captures rotational and positional invariant features.

Pooling: There are several pooling operations but the most common is max pooling. It teaches the network spatial variance, in simple words the ability to recognize the image features even if the image is upside down, tilted, the image is taken from far or close, etc. The output of this operation is a pooled feature map.

Flattening: The purpose of this operation is to be able to input the pooled feature map into the neural network. The below image shows the entire CNN operation.

6.29 What is Epoch, Batch, and Iteration in Deep Learning?

- Epoch is a one-pass to the entire dataset. Here, one pass = one forward pass + one backward pass
- Batch size is the number of training examples in one forward/backward pass. The larger batch size requires more memory space.
- Iteration is the number of passes. If each pass using batch size then the number of times a batch of data passed through the algorithm.
- For example, if you have 1000 training examples, and your batch size is 200, then it will take 5 iterations to complete 1 epoch.

6.30 What is an Auto-encoder?

Autoencoders are unsupervised deep learning techniques that reduce the dimension of data to encode. Autoencoders encoded the data on one side and decoded it on another side. After encoding, it transforms data into a reduced representation called code or embedding(also known as latent-space representation). This embedding then transformed into the output. Autoencoders can do dimensionality reduction and improve the performance of the algorithm.

6.31 What do you understand by Boltzmann Machine and Restricted Boltzmann Machines?

Boltzmann machines are stochastic(non-deterministic models) and generative neural networks. It has the capability to discover interesting features that represent complex patterns in the data. Boltzmann Machine uses many layers for feature detectors that make it a slower network. Restricted Boltzmann Machines (RBMs) have a single layer of feature detectors that makes them faster compared to Boltzmann Machine.

RBM is a neural network model are also known as Energy-Based Models. RBM offers various applications in recommender systems, classification, regression, topic modeling, and dimensionality reduction.

6.32 Explain Generative Adversarial Network.

GAN (Generative Adversarial Network) is unsupervised deep learning that trains two networks at the same time. It has two components: Generator and Discriminator. The generator generates the images close to the real image and the discriminator determines the difference between fake and real images. GAN is able to produce new content.

6.33 What is Adam? What's the main difference between Adam and SGD?

Adam (Adaptive Moment Estimation) is a optimization technique for training neural networks. on an average, it is the best optimizer .It works with momentums of first and second order. The intuition behind the Adam is that we don't want to roll so fast just because we can jump over the minimum, we want to decrease the velocity a little bit for a careful search. Adam tends to converge faster, while SGD often converges to more optimal solutions. SGD's high variance disadvantages gets rectified by Adam (as advantage for Adam).

6.34 When would you use Adam and when SGD?

Adam tends to converge faster, while SGD often converges to more optimal solutions.

6.35 Neural Network: How Do They Learn

The learning happens by passing labeled data all the way from the input layers to the output layers then all the way back. Since the data is labeled the Neural network knows what's the expected output and compares it to the actual output of the Neural Network.

In the first Epoch, the labeled data is entered at the input layer and propagated to the output layer where your Neural Network will calculate an output. The difference between the actual output of your Neural Network vs. the expected output is called the Cost Function. The goal of your Neural Network is to decrease this Cost Function as much as possible. So, your Neural Network will Back-Propagate from the output layer all the way to the input layer and update the weights of the Neurons accordingly in an attempt to minimize this Cost Function.

The act of sending the data from the input layer to the output layer then all the way back is called an Epoch. In each epoch, the Neural Network updates the weights of the Neurons which is also known as Learning. After multiple Epochs and weight updates, the loss function (the difference between Neural Network output vs. Actual output) should reach a minimum.

6.36 Neural Network during Learning Phase

Upon learning, the Neurons will have different weights and those weights will dictate future outputs. For example in our earlier car vs. bus classification scenario a Neuron could be looking at the number of windows to decide if the object is a car or bus, obviously, this Neuron will have higher weight than a Neuron looking at the color of the object to determine if it's a car or a bus. This is an oversimplification of the Neurons function but I want you to get the idea of Neuron weights according to importance.

6.37 Gradient Descent

Gradient Descent is a method to minimize the Cost Function in order to update the Neurons weights. This method tells your Neural Network how to calculate the Cost Function in a fast efficient manner to minimize the difference between the actual and expected outputs.

The easiest to understand and most common example is comparing your Cost Function to a ball trying to find the lowest point by updating its slope.

Gradient (Batch) Descent when your Neural Network goes through the data one row at a time and calculate the actual output for each row. Then after finishing all rows in your dataset, the Neural Network compares the cumulative total output of all rows to the expected output and backpropagates to update the weights. This means the Neural Network updates the weights once after working through the entire dataset as one big batch, That's a big timely Epoch. The Neural Network will do this several times to train the network.

Stochastic Gradient Descent when your Neural Network goes through the data one row at a time and calculate the actual output for each row. Right away the Neural Network compares the actual output of the first row to the expected output and backpropagates to update the weights, that completes an Epoch. Then the same happens for the second row, comparing outputs and backpropagating to update weights. All the way to the last row, so multiple Epochs and multiple weight updating happen to go through the entire dataset rather than treating it as one big batch like in the case of Gradient Descent. This helps to avoid local minimums and it's faster than Gradient Descent cause it doesn't need to load all data in memory and run through them at once rather it loads one row at a time and updates weights.

There is the best of both worlds method that's called mini-batch Gradient Descent which is basically combining both. You decide how many rows to run and update at once. So instead of running the whole dataset as one batch or instead of running one row at a time, you have the flexibility to choose any number of rows to run.

6.38 Back-Propagation

By now you should know what back-propagation is if you don't then it's simply adjusting the weights of all the Neurons in your Neural Network after calculating the Cost Function. Back-Propagation is how your Neural Network learns and its the result of calculating the Cost Function. The important concept to know is that Back-Propagation updates all the weights of all the Neurons simultaneously.

For training purposes, in the beginning, the weights of the Neurons are randomly initialized with small numbers then through learning and back-propagation the weights start to get updated with meaningful values.

6.39 Softmax Function

This is mainly used in classification problems for multi-class predictions. It is typically in the output layer of the Neural Network.

You could apply different activation functions for hidden layers and output layers. In this diagram, ReLU is applied for hidden layers while Sigmoid is applied for the output layer. This is common to predict the probability of something.

6.40 What happens if the learning rate is set too high or too low?

If the learning rate is too low, your model will train very slowly as minimal updates are made to the weights through each iteration. Thus, it would take many updates before reaching the minimum point.

If the learning rate is set too high, this causes undesirable divergent behavior to the loss function due to drastic updates in weights, and it may fail to converge.

Chapter 7

SQL

We have finished a nice book.