FIT5202 Data processing for big data¶

Activity: Machine Learning with Spark (Classification Using Decision Tree, Random Forest, and Logistic Regression)¶

Last week we learnt about basics of machine learning with Apache Spark. **`MLlib`** is Apache Spark's scalable machine learning library. Its goal is to make practical machine learning scalable and easy. At a high level, it provides tools such as:

- ML Algorithms: common learning algorithms such as classification, regression, clustering, and collaborative filtering
- Featurization: feature extraction, transformation, dimensionality reduction, and selection
- Pipelines: tools for constructing, evaluating, and tuning ML Pipelines
- Persistence: saving and load algorithms, models, and Pipelines
- Utilities: linear algebra, statistics, data handling, etc.

We looked into transformers, estimators and machine learning pipeline in the last weeks tutorial activity.

This week we have learnt about different classification algorithms in the lecture. We will look into how to use the different popular family of classification and regression methods; Decision Trees and Random forests.

Table of Contents¶

Bank case study: Will the client subscribe? ¶

The data was used to direct marketing campaigns of a banking institution. The marketing campaigns were based on phone calls. The classification goal is to predict whether the client will subscribe (1/0) to a term deposit.

Attributes¶

1. age (numeric)
2. job : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')
3. marital : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)
4. education (categorical: 'primary', 'secondary', 'tertiary', 'unknown')
5. default: has credit in default? (categorical: 'no','yes','unknown')
6. balance : bank balance
7. housing: has housing loan? (categorical: 'no','yes','unknown')
8. loan: has personal loan? (categorical: 'no','yes','unknown')

Related with the last contact of the current campaign:¶

1. contact: contact communication type (categorical: 'cellular','telephone','unknown')
2. day: last contact day of the week (numerical: 1,2,...28,29,30)
3. month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
4. duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

Other attributes:¶

1. campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)

2. pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
3. previous: number of contacts performed before this campaign and for this client (numeric)

4. poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success','unknown')

Output variable (desired target):¶

1. deposit - has the client subscribed a term deposit? (binary: 'yes','no')

1. Lab Task: Complete the following steps for the initial part to get the data ready for the Classification Algorithms. 1. **Loading data:** Load the `bank.csv` file using spark session 2. **Prepare the data :** Prepare data for machine learning and preprocess the data according to the algorithm for training. 3. **Feature Engineering:** Use `StringIndexer`, `OneHotEncoder` and `Vector Assembler` to transform the dataset into *features* and *label* columns. 4. **Pipeline API :** Assemble the above steps of transformation into a `Pipeline`. Use the pipeline to **transform** the data. 5. **Train/Test Split :** For the transformed data, create a **train/test** split of **80%/20%**.

Step 1: Data Loading and Preparation ¶

In this step, you can do some data exploration like:

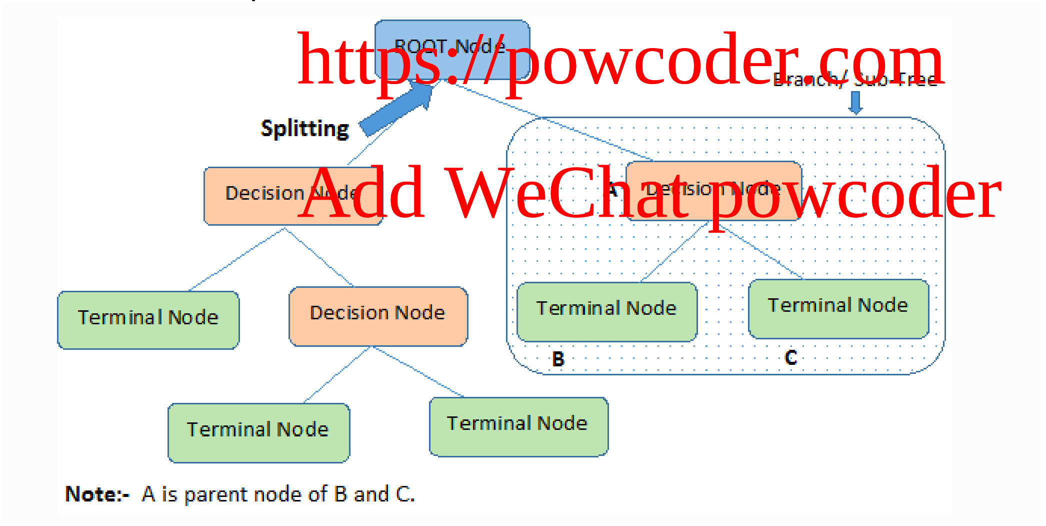
1. See some sample data, check the schema
2. Check the statistics of the numerical columns in the dataset
3. Target Variable Distribution, what are the number of instances the target variable has?
4. Check if the dataframe contains null values?
5. Separate the numerical and non-numerical columns to apply feature engineering

In [ ]:
```
#Write your code here
```
Step 2: Feature Engineering ¶
Hint : Use all the features, separate the numerical and non-numerical features to simplify the Feature Engineering process. For the **target variable, make sure 1 is for Yes and 0 is for No** while converting it to numeric representation.

In [ ]:
```
#Write your code here
```
Step 3: Pipeline API ¶
In [ ]:
```
#Write your code here
```
Step 4: Train/Test Split ¶
Hint : Use seed while splitting data to identify different sets of train/test splits.

In [ ]:
```
#Write your code here
```
ML Classification Models ¶

Decision Tree ¶
Decision tree algorithms are said to be widely used because they process categorical data and are readily available in classification tasks by multiple classes. The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by learning simple decision rules inferred from prior data(traing data). The picture below shows some components in a decistion tree.



Decision trees use multiple algorithms to decide to split a node into two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that the purity of the node increases with respect to the target variable. The decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.

On the downside, decision trees are prone to overfitting. They can easily become over-complex which prevents them from generalizing well to the structure in the dataset. In that case, the model is likely to end up overfitting which is a serious issue in machine learning. To overcome this decision tree hyperparameters could be tuned. For a description of some parameters, refer to this link

In [ ]:
```
from pyspark.ml.classification import DecisionTreeClassifier

# Extracts the number of nodes in the decision tree and the tree depth in the model
```

```
and stores it in dt.
dt = DecisionTreeClassifier(featuresCol = 'features', labelCol = 'label', maxDepth =
3)
dtModel = #WRITE CODE : Use the fit method to train the model with the training data
you created in Step 4
```

In [ ]:

```
dtPredictions = #WRITE CODE to get predictions from the test data
#WRITE CODE to Display the predictions
```

NOTE: You can see that DecisionTree has a parameter called `maxDepth`. Discuss the significance of this parameter.

Random Forest ¶

In [ ]:

```
# WRITE CODE: Implement Random Forest Classifier
from pyspark.ml.classification import RandomForestClassifier
#WRITE CODE : Create a Random Forest Classiication model and train it
```

In [ ]:

```
#WRITE CODE : get the predictions for the test data
```

Logistic Regression ¶

Logistic Regression is a classification algorithm. It is used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables. To represent binary/categorical outcome, we use dummy variables. You can also think of logistic regression as a special case of linear regression when the outcome variable is categorical, where we are using log of odds as dependent variable. In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function.

In [ ]:

```
# TODO: Implement Logistic Regression Classifier
from pyspark.ml.classification import LogisticRegression

#WRITE CODE Create an initial model using the train set.
```

In [ ]:

```
#WRITE CODE: Write the Predictions for test data and display the predictions
```

NOTE: Discuss various parameters used in `Logistic Regression`.

Model Evaluation ¶

Our goal is not just to build a model, it is about selecting a model which gives high accuracy on our sample data. There are different kinds of evaluation metrics, the choice of these depend on the type and implementation plan. More details on evaluation metrics for classification in Spark can be found here. We are going to cover the following evaluation in this session:

- Confusion Matrix
  - Accuracy
  - Precision
  - Recall
  - F1-Score
- AUC-ROC

Accuracy¶

Accuracy is useful when the target class is well balanced but is not a good choice with unbalanced classes. For example, if a model is designed to predict fraud from a dataset where 95% of the data points are not fraud and 5% of the data points are fraud, then a naive classifier that predicts not fraud, regardless of input, will be 95% accurate. For this reason, metrics like precision and recall are typically used because they take into account the type of error. In most applications there is some desired balance between precision and recall, which can be captured by combining the two into a single metric, called the F-measure.

Confusion Matrix ¶

Confusion Matrix is a performance measurement for machine learning classification problem where output can be two or more classes. It is a table with 4 different combinations of predicted and actual values as shown in the picture below where T/F refers to true and positive respectively, and P/N to positive and negative respectively.

From the confusion matrix, we can obtain the following indicators.

- **Recall:** Out of all the positive classes, how much we predicted correctly. It should be high as possible.
- **Precision:** Out of all the positive classes we have predicted correctly, how many are actually positive.
- **Accuracy:** Out of all the classes, how much we predicted correctly. It should be high as possible.
- **F1-Score : Combining Precision and Accuracy:** It is the weighted average of Precision and Recall. It takes both **False Positives** and **False Negative** into account.

2. Lab Task: Calculate the remaining **False Negative and False Positive** for `Decision Tree` based on the example below. Then Compute the other metrics, i.e. **Accuracy**, **Precision**, **Recall** and **F1-Score** for the DT Model. **NOTE:** In our case we are using Yes=1 and No=0. Remember that 1 doesn't mean positive always, it will depend on how the Target variable is encoded as well.

In [ ]:

```
# Calculate the elements of the confusion matrix
TN = dtPredictions.filter('prediction = 0 AND label = 0').count()
TP = dtPredictions.filter('prediction = 1 AND label = 1').count()
FN = #WRITE CODE to find the False Negative
FP = #WRITE CODE to find the False Positive

# show confusion matrix
dtPredictions.groupBy('label', 'prediction').count().show()
# calculate metrics by the confusion matrix
accuracy = #WRITE CODE : formula to find accuracy
precision = #WRITE CODE : formula to find precision
recall = #WRITE CODE : formula to find recall
f1 = #WRITE CODE : formula to find F1-score

#WRITE CODE : Display the various metrics calculated above
```

3. Lab Task: Create a function `compute_metrics()` which takes **predictions** as input parameter and computes all these metrics. Use the function to compute the 4 metrics for all 3 Classification Algorithms.

In [ ]:

```
def compute_metrics(predictions):
    #WRITE CODE: to calculate accuracy,precision,recall and f1 based on above example
    return accuracy,precision,recall,f1
```

In [ ]:

```
#WRITE CODE : Print the accuracy,precision,recall and f1 scores for each
Classification algorithm, using the function created
```

4. Lab Task: Present the accuracies, precision and recall of the different classification algorithms in a bar chart using `matplotlib`. You can use the function given here:

```
    import matplotlib.pyplot as plt
    %matplotlib inline
    def plot_metrics(x,y):
        plt.style.use('ggplot')
        x_pos = [i for i, _ in enumerate(x)]
        plt.bar(x_pos, y, color='blue')
        plt.xlabel("Classification Algorithms")
        plt.ylabel("AUC")
        plt.title("Accuracy of ML Classification Algorithms")
        plt.xticks(x_pos, x)
        plt.show()
```

What do you observe about the differences in the accuracy/precision/recall/f1-scores for these different classification algorithms? How do you decide the most suitable metric? Discuss this with your tutor.

Area Under the Curve (AUC-ROC) ¶

When we need to check or visualize the performance of the multi - class classification problem, we use AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve. It is one of the most important evaluation metrics for checking any classification model's performance.

AUC - ROC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s. In the example below, we are using Spark's `BinaryClassificationEvaluator` to compute the AUC-ROC curve.

Binary Classification Metrics ¶

Now let's evaluate the model using BinaryClassificationMetrics class in Spark ML. BinaryClassificationMetrics by default uses areaUnderROC as the performance metric. Read More

5. Lab Task: Compute the `Area Under ROC` for the two other Algorithms based on the example below.

In [ ]:

```
# Use BinaryClassificationEvaluator to evaluate a model
from pyspark.ml.evaluation import BinaryClassificationEvaluator

# Evaluate model Decision Tree
evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
auc_dt = evaluator.evaluate(dtPredictions)
print(auc_dt)
print(evaluator.getMetricName())
```

In [ ]:

```
#WRITE CODE : area under curve for Random Forest
```

In [ ]:
```
#WRITE CODE : area under curve for Logistic Regression
```
Visualizing AUC-ROC ¶

We can easily visualize the ROC curve for **Logistic Regression** using the `BinaryClassificationEvaluator`. The example below shows how to plot it using matplotlib.

In [ ]:
```
import matplotlib.pyplot as plt

print("Area Under ROC: " + str(evaluator.evaluate(lrPredictions,
{evaluator.metricName: "areaUnderROC"})))

# Plot ROC curve
trainingSummary = lrModel.summary
roc = trainingSummary.roc.toPandas()
plt.plot(roc['FPR'],roc['TPR'])
plt.ylabel('TPR')
plt.xlabel('FPR')
plt.title('ROC Curve')
plt.show()
```
However, for other algorithms, we need to calculate it manually. The ROC Curve is simply a line plot of FPR and TPR across all thresholds (i.e. 0-1). We can calculate TPR and FPR from the confusion matrix. However, by default, a threshold value of 0.5 is used while computing the confusion matrix. So we need a way to compute the confusion matrix for different thresholds.

Getting the thresholds¶

We can simply use numpy `linspace` to get a list of thresholds.

In [ ]:
```
import numpy as np
print(np.linspace(0, 1, 100))
```

If we look at the `probability` column of the predictions, we will see it is an `Array` with two probability values, the first one for negative class and the second one for positive class. Now we can consider the **probability for positive class** to decide which value is **positive** and which value is **negative**

For example, if our threshold is 0.7, we can say **if the positive probability is greater than 0.7, it is a positive prediction else it is a negative prediction**.

Here is an example for the Decision Tree Predictions.

In [ ]:
```
#User Defined Function to split the probabilities into two columns
import pyspark.sql.functions as F
import pyspark.sql.types as T
to_array = F.udf(lambda v: v.toArray().tolist(), T.ArrayType(T.FloatType()))
```
In [ ]:
```
#Splitting the probability to 2 parts using the UDF
df = dtPredictions.withColumn('probability', to_array('probability'))
```
In [ ]:
```
#A new df which contains the probabilites in separate columns
prob_df =
df.select(df.probability[0].alias('negative_prob'),df.probability[1].alias('positive_
prob'),'label')
```
Now, we have extracted the probabilities for positive and negative class. Now we need to create

our own `prediction` column based on a threshold we are provided.

In [ ]:

```
#Here based on the threshold, the prediction column is computed
threshold=0.7
prob_df.withColumn('prediction',F.when(prob_df.positive_prob >
threshold,1).otherwise(0))
```

Now we can create the confusion metrics based on the new predicted class. Let's create a function to do that. The function returns the TN,TP,FN and FP values.

CHALLENGE TASK: the following confusion_matrix function is slow, why is it so slow? Can you refactor it?

In [ ]:

```
def confusion_matrix(predictions):
     # Calculate the elements of the confusion matrix
    TN = predictions.filter('prediction = 0 AND label = 0').count()
    TP = predictions.filter('prediction = 1 AND label = 1').count()
    FN = predictions.filter('prediction = 0 AND label = 1').count()
    FP = predictions.filter('prediction = 1 AND label = 0').count()
    return TP,TN,FP,FN
```

In [ ]:

```
#TESTING
#for threshold 0.7, lets calculate the TN,TP,FN,FP from confusion matrix
threshold=0.7
test_df=prob_df.withColumn('prediction',F.when(prob_df.positive_prob >
threshold,1).otherwise(0))
tp,tn,fp,fn = confusion_matrix(prob_df)
tpr = tp/(tp+fn)
fpr = fp/(fp+tn)
print('TPR:',tpr,'FPR:',fpr)
```

CHALLENGE TASK: Based on all the above information, create a function, which loops through all the thresholds and computes the TPR and TFR for each threshold. Store all the TPR and TFR values in separate arrays and later visualize the **ROC Curve** for **Decision Tree** and **Random Forest**.

In [ ]:

```
#WRITE CODE
```

Congratulations on finishing this activity. See you next week.¶

References¶

1. https://www.kdnuggets.com/2020/04/performance-evaluation-metrics-classification.html
2. https://spark.apache.org/docs/latest/mllib-evaluation-metrics.html