Sneha Wattamwar - G01209653
Devika Walavalkar - G01211392

CS 657 Mining Massive Datasets

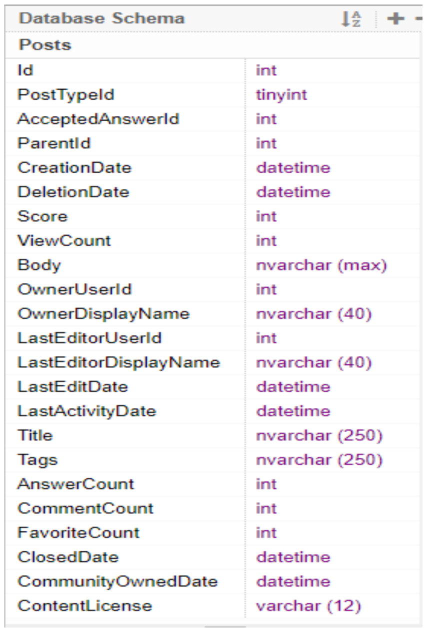
Project: Pyspark project using StackOverflow Data

Introduction: Quoted from the StackOverflow website, '*Stack Overflow is an open community for anyone that codes. We help you get answers to your toughest coding questions, share knowledge with your coworkers in private, and find your next dream job*.' The performance of this website can reach its full potential depending on how well we use tags. It is observed that tags of posts/questions are critical to enhancing the user experience which in turn impacts its turnover. Our proposed project attempts to complete tasks using this data to implement the knowledge gained from the Mining Massive Datasets course. The project comprises of two tasks, namely, predicting tags of questions and predicting the score a question will be receiving.

Significance of the two tasks:

- Task1: When a question is posted on StackOverflow, either the user provides tags for it or StackOverflow uses its algorithm to predict these tags. These tags are of critical significance as the accuracy of predicting them makes sure that a particular question is not lost or left unanswered in the enormous pool of questions. They are used as triggers to notify users who have expertise in the respective domain and have a good reputation of providing effective solutions.
- Task2: The score of a question is calculated by subtracting the number of downvotes from the number of upvotes it receives. This score is applied to its tags to assign a reputation score to any user answering the question and which in turn is a measure of the amount of knowledge a user has in that domain. A reputation score threshold is also used to provide users with offers and access to special features of StackOverflow if they have a reputation score greater than the threshold.

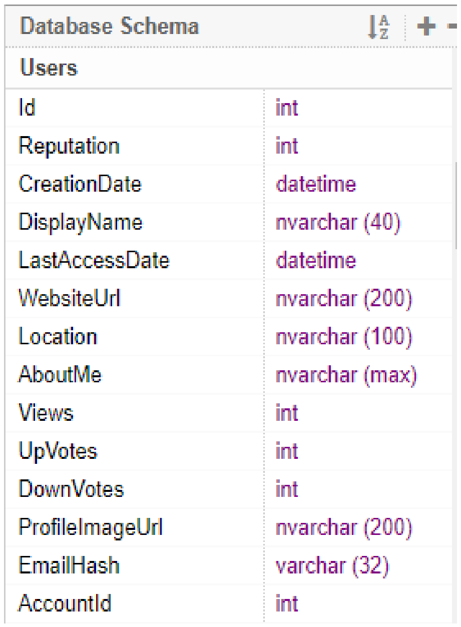
Data Source: Stack Exchange releases "data dumps" of all its publicly available content roughly every three months via archive.org, and also makes that information queryable over the Internet at the Stack Exchange Data Explorer (SEDE). We used this publicly available StackOverFlow data for our project. Following screenshots displays the tables used along with its schema.



Database Schema

Posts	
Id	int
PostTypeId	tinyint
AcceptedAnswerId	int
ParentId	int
CreationDate	datetime
DeletionDate	datetime
Score	int
ViewCount	int
Body	nvarchar (max)
OwnerUserId	int
OwnerDisplayName	nvarchar (40)
LastEditorUserId	int
LastEditorDisplayName	nvarchar (40)
LastEditDate	datetime
LastActivityDate	datetime
Title	nvarchar (250)
Tags	nvarchar (250)
AnswerCount	int
CommentCount	int
FavoriteCount	int
ClosedDate	datetime
CommunityOwnedDate	datetime
ContentLicense	varchar (12)

Fig1



Database Schema

Users	
Id	int
Reputation	int
CreationDate	datetime
DisplayName	nvarchar (40)
LastAccessDate	datetime
WebsiteUrl	nvarchar (200)
Location	nvarchar (100)
AboutMe	nvarchar (max)
Views	int
UpVotes	int
DownVotes	int
ProfileImageUrl	nvarchar (200)
EmailHash	varchar (32)
AccountId	int

Fig2

We will be cleaning, transforming, and using the columns as features which contribute the most to our target predictions. For our classification models to predict tags, the two most important columns are the 'Title' and 'Body'. The title is usually a one line or couple of lines of question statement and gives an abstract information of the problem statement. The body further contains the relevant words which are used to formulate the description of the question. When downloaded, both the fields are HTML formatted text which needs to be pre-processed. The 'PostTypeId' column is used to filter-out posts which are 'Answers' and not Questions. (1-Question, 2-Answer). For our task 2 which uses regression models, we will be predicting the score of a question that has been posted. In this case, we will require more columns which contribute to the prediction. We will be using the columns relating to a user posting the question from the Users table such as 'OwnerUserId', 'Reputation', 'UpVotes' and 'DownVotes'. From the Posts table, we will be using attributes such as 'Score', 'AnswerCount', 'FavouriteCount', 'ViewCount' and 'Tags'. We will be further seeing in the report why we choose these attributes.

Approach for Task 1:

- Prepare the data
  1. Download the data from Kaggle/Scrape data from StackExchange through SQL queries.
  2. Create a SparkSession, sparkContext and a SQLContext and read all the data into a spark dataframe.
  3. Select the columns 'Id', 'PostTypeId', 'Title', 'Body' and 'Tags' from the dataframe.
  4. Filter out all the posts of the Answers category using the 'PostTypeId' column and drop the column.
  5. Filter out rows with tags of count less than a threshold for creating uniformity in data. In our project the count threshold is 10,000. Hence, we remove outlier classes.
  6. Filter out rows with no tags or 'null' tags and convert data frame into rdd.
  7. Further, the title and the body_text is cleaned in parallel using map function of the rdd. The clean_text function is called for every row. Remove HTML tag <code> from the body content, convert text to lower, remove punctuations, remove digits, remove stopwords and return an array of words.
  8. Clean the tag column by stripping off '<' and '>'
  9. Concatenate the title and body text.
  10. Now, we have 3 columns, Id, Text and Tags.
  11. Convert it into a dataframe for further implementation.

- Feature Extraction and Feature Transformation
  1. As this is a multiclass classification problem, the tags (datatype 'string') is converted into float using the StringIndexer library to encode the string column into a column of label indices. Every index represents a 'tag' and this column will be our labels for training the models.
  2. The text column is vectorized and relevant features are extracted from the text column. Thus, we perform fit and transform on the dataframe using CountVectorizer. The CountVectorizer tokenizes the collection of documents and builds a vocabulary of known words using fit, and encodes the text using that vocabulary. The parameters passed are vocabSize = 10000 and minDF = 5. The minDF parameter ignores terms that have a document frequency strictly lower than the given threshold when building the vocabulary as they are comparatively insignificant to the other terms.
  3. Further, we calculate the TF-IDF of the terms which will be fed into the classification model as features. This is a measure of relevance of every word to the tag.
  4. The data is now ready to be split into train and test for training the models and evaluating the performances.

- Training and evaluating different Multiclass classification models
  1. Split the data into 80% train -20% test ratio for model performance evaluation.
  2. In this project we will be training three classification models using the train set.
  3. The tags for the test set are predicted using these trained models.
  4. The metric for evaluation is F1 score (precision+recall) which is provided by the MulticlassClassificationEvaluator library.
  5. This evaluation metric is chosen because if we predict an incorrect tag then Precision will decrease and if we missed out an important tag then Recall will decrease. Thus, F1 score is used as it only gives good value if both the Precision and Recall are high. The accuracy of Stackoverflow tag prediction increases for every post/question that is assigned the correct tag.
  6. These models compare every test row for the class that it fits in the best and outputs the class as the prediction.

- Build an algorithm to suggest/predict tags
  1. In addition to Task1, we implemented an algorithm which suggests the user 3 additional tags apart from the primary tag.
  2. We create an rdd with the post Id and the TF-IDF vector that we obtain in the previous steps.
  3. For every row i.e. for every post/question, we send the vector to the function getIndices ().
  4. It sorts the vector in the descending order and returns the index of the top 3 values. It indicates that the words in the vocabulary mapped to those 3 indices having the highest 3 TF-IDF values describe the tag the most.
  5. Map the indices to its words in the vocabulary and return the suggestions/predictions.

Approach for Task 2:

- Prepare the data
  1. Select columns 'Id0', 'Tags', 'ViewCount', 'OwnerUserId', 'AnswerCount', 'CommentCount', 'FavoriteCount', 'Reputation', 'Views', 'UpVotes', 'DownVotes', 'Score' which we will be using for prediction of scores of posts. The column ‘Score’ is the labels for training the regression models.
  2. Filter out post/ questions with null or no tags.
  3. Fill all the columns with null values with 0. We now have all the feature columns with datatype int/float except the column ‘Tags’.

- Feature Extraction and Feature Transformation
  1. The tags (datatype ‘string’) is converted into float using the StringIndexer library to encode the string column into a column of label indices where very index represents a ‘tag’.
  2. Using the VectorAssembler library, convert the list of all the feature columns into a single vector column.
  3. Use PCA for feature dimensionality-reduction to reduce the 10-dimensional feature vectors into 3-dimensional principal components. The reduced set still contains most of the information as the original one.

- Training and evaluating different Regression models
  1. This step is similar to the one in Task 1. Split the data into train and test set. As this is a Regression problem, we will be training different regression models using the training data and predicting scores for the records in the test set.
  2. The scores compared for their performances using the RMSE (Root Mean Square Error) and MAE (Mean Absolute Error) metric.

3. RMSE is a better evaluation metric for our data as the labels have both positive and negative values. MAE calculates using the absolute values. RMSE penalizes more if the difference between the label and prediction is large which give us an idea of how poor or good does a model perform.

Results / Observations:

1. Observations on Task 1: As we can see that the One-vs-Rest Classifier, which is a modified version of Logistic Regression under the hood, performs the best on our multiclass data because it considers binary values for different classes and chooses the one with highest probability. Naïve Bayes performs good too because when encountering a missing value, the algorithm omits that property in its calculations, which is only applicable due to the assumption that all attributes are independent. However, on the other hand, Random Forest Classifier performs very poorly. It is undoubtedly a very good classification model but however, it is not a hidden fact that it performs poorly with huge sparse data. The reason being that it selects nodes at various levels and as the data is sparse, there is a high chance that it selects 0 as the nodes for splitting. This results in incorrect classification and thereby not so good results.

Multi-class Classification models: F1 Score (Precision + Recall)

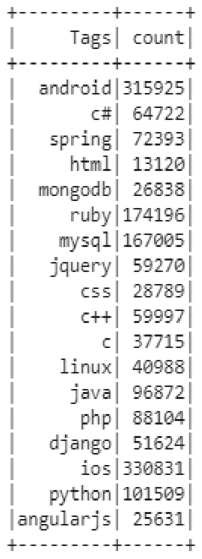
Naïve Bayes Classifier	Random Forest Classifier	One-vs-Rest Classifier
0.85	0.18	0.92

2. Observations on Task 2: All the three Regression models perform good on our multiclass data as we can see the RMSE and MAE values are below. As our prediction labels range from -25 to +2037, the RMSE and MAE values in this range seems to be acceptable.

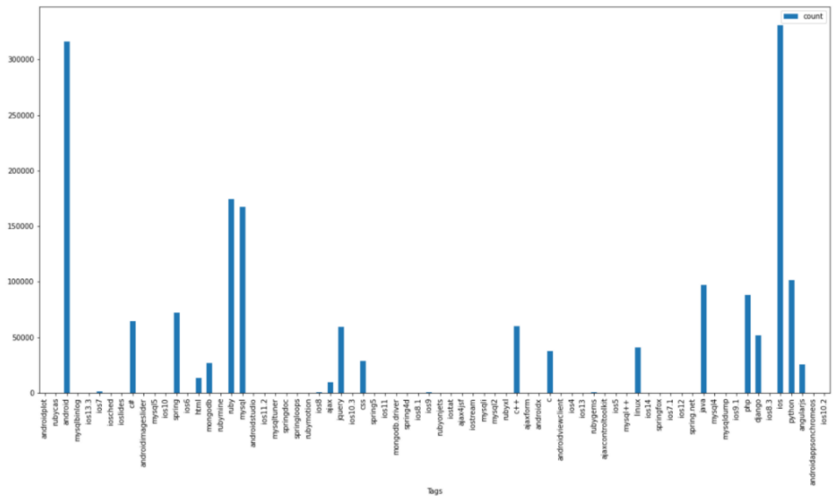
Regression models:

	Linear Regressor	Random Forest Regressor	Gradient-Boosted tree Classifier
RSME	5.19	6.68	6.30
MAE	1.3	1.4	1.41

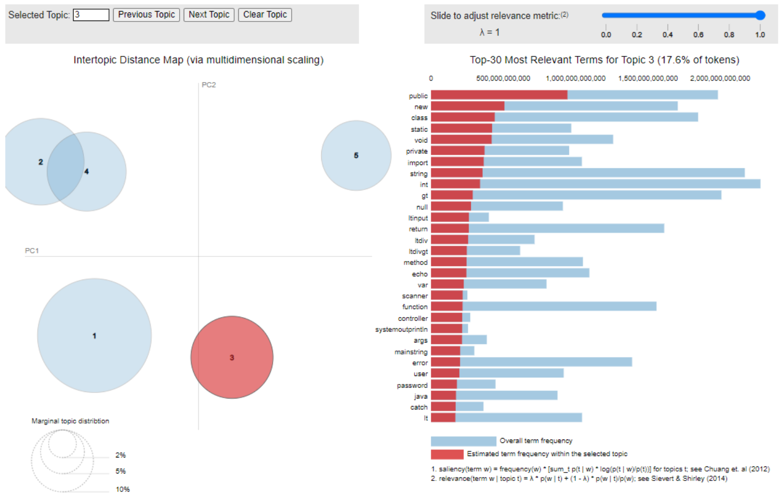
3. The following bar graph on the right displays the number of records for each tag. As we can observe, the data is not uniform and therefore, we filter out records of particular tags using a threshold. On the left is the final count and names of tags that will our class labels.



```
+---------+------+
|     Tags| count|
+---------+------+
|  android|315925|
|       c#| 64722|
|   spring| 72393|
|     html| 13120|
|  mongodb| 26838|
|     ruby|174196|
|    mysql|167005|
|   jquery| 59270|
|      css| 28789|
|      c++| 59997|
|        c| 37715|
|    linux| 40988|
|     java| 96872|
|      php| 88104|
|   django| 51624|
|      ios|330831|
|   python|101509|
|angularjs| 25631|
+---------+------+
```



4. We have provided an Interactive visualization using the LDA model and the pyLDAvis tool. It displays the relevance of words with a particular tag and can be used to explore data. For example, the highlighted circle is 'JAVA' and we can see all the related keywords on the right.



Conclusion:

The tags of StackOverflow are closely related to the title and body of the questions. Every programming language has its own set of words, syntactical and semantical, which helps to distinguish between programming languages. We explored classifiers and regressors apart from the ones covered in class and analyzed its results. Computing of prediction of labels with multiple values is expensive than binary classification as every class is evaluated for every prediction to output the class with highest probability. Our best model being the One-vs-Rest multiclass classifier has F1 score of 0.92 which is pretty good, considering the size of data we are dealing with. Also, for regression, on an average, all three models were able to predict the Scores fairly after performing feature reduction.