Predicting Tag of Questions & Data Science Job Required Skill Analysis

STEVENS INSTITUTE OF TECHNOLOGY, SPRING 2017, CS - 513

Projects By:

Priya Parmar

10412380

Ruchika Sutariya

10418975

Harsh Kevadia

10420312

Agenda

- Problem Statement
- Objective
- Project Flow
- Data Scraping
- DataSet After Scraping
- Cleaning Data
- Classification Models
- Conclusion

What is Career Cup ?

- CareerCup helps people prepare for jobs at tech companies.
- Unlike other types of interviews, technical interviews are intensely skill based.
- So what CareerCup does -- it offers you ways of studying for an interview.
  - You can ask questions for the interview prep.
  - You can post questions that were asked to you in your interview to help others.

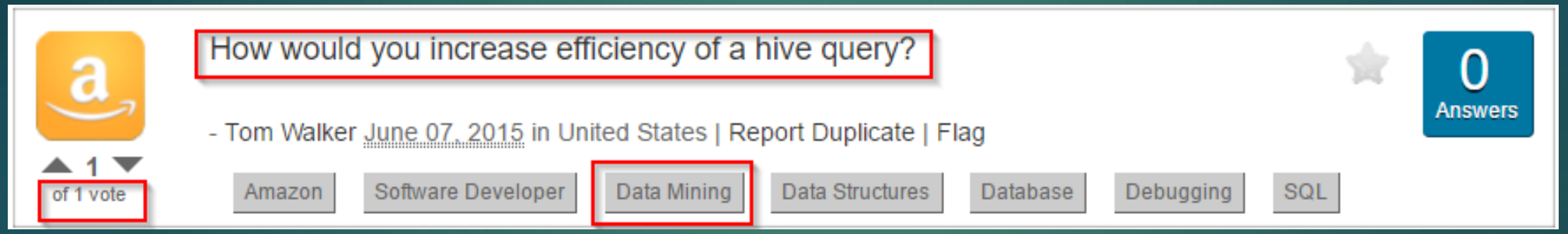
Problem Statement

- Some users don't put tags in their questions.
- This leads to questions with ambiguous categories.
- It becomes a cumbersome process to decide what questions belong to which category manually leading to increase in human work load.
- There has to be a way to categorize these questions that don't have tags/categories.

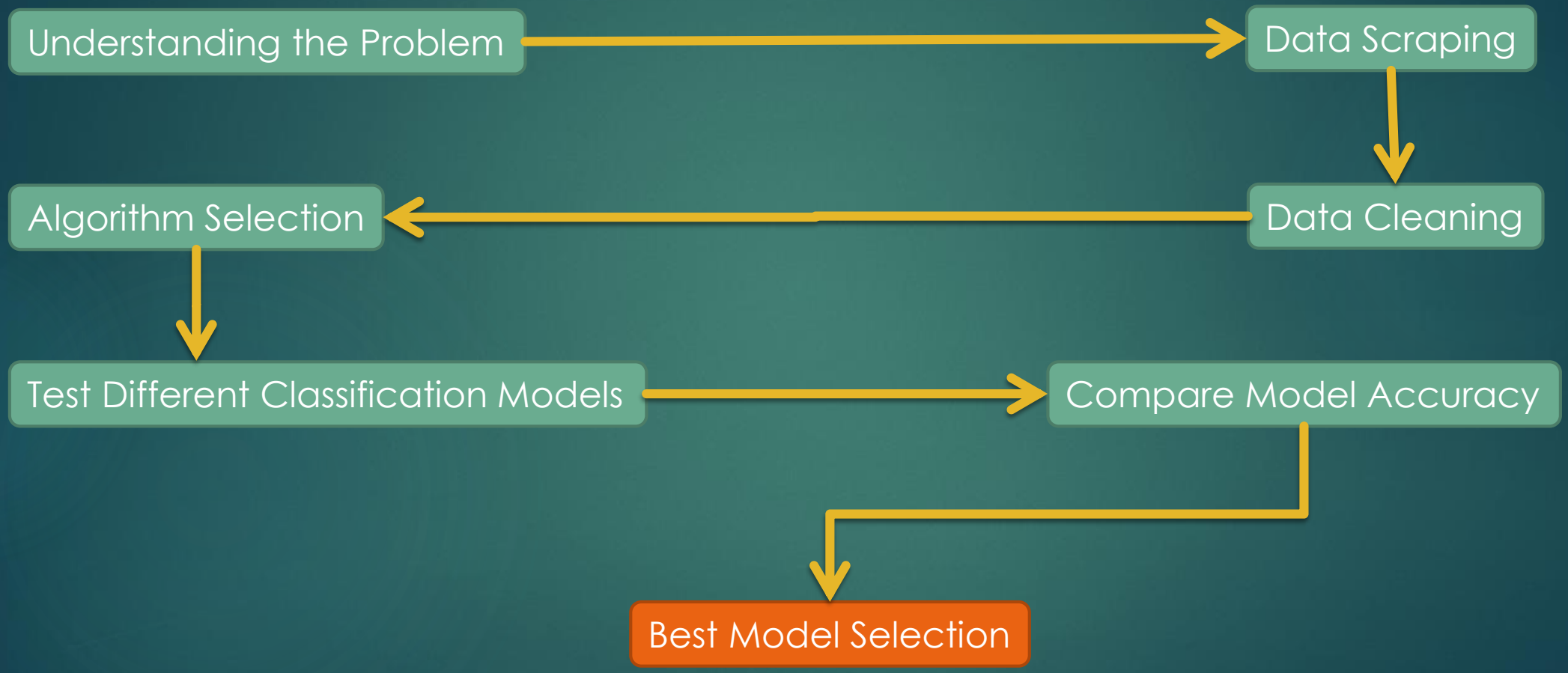
Objective

- We are focusing on predicting category of questions based on their properties.
- We are considering previous questions, their votes and their tags for the categorization of those questions.
- The main aim of this project is to predict the category of questions by using the previously categorized questions leading to less human work load.

An Example ..


How would you increase efficiency of a hive query?
- Tom Walker June 07, 2015 in United States | Report Duplicate | Flag
1
of 1 vote
0
Answers
Amazon
Software Developer
Data Mining
Data Structures
Database
Debugging
SQL

Project Flow



Data Scraping

- Web sites are written using HTML, which means that each web page is a structured document.
- Data scraping is the practice of using a computer program to sift through a web page and gather the data that you need in a format most useful to you while at the same time preserving the structure of the data.
- So we used a Python program to scrap through the CareerCup website to get a real time dataset.

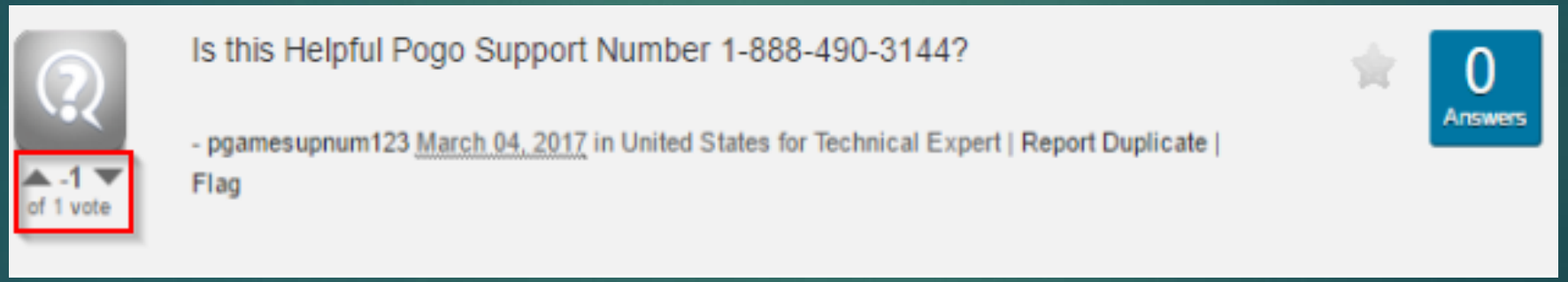
After Scraping Data

- The format that we get after scraping:
  - TAG \t VOTE \t Question
- Example:

- algorithm 3 Given the root of a Binary Tree along with two integer values. Assume that both integers are present in the tree. Find the LCA (Least Common Ancestor) of the two nodes with values of the given integers. 2 pass solution is easy. You must solve this in a single pass.

Cleaning Data

- Removing the questions with negative votes to improve the quality of the dataset.
- For each question, we use text mining to remove stop words (like is, the, etc) from the question.



Process - Convert Raw Questions to the matrix

	Word 1	Word 2	Word 3
Algorithm	1	3	0
PHP	4	0	1
Hash Table	0	1	2
Trees and Graphs	0	5	0

Extracting words

- Words are extracted in two ways:
  1. Question is split by spaces to get individual words.
  2. A fixed length of 5 characters is used to define a word.

Classification Models Used

1. KNN
2. Decision Tree
3. Logistic Regression
4. Random Forest
5. Naive Bayesian
6. ANN

KNN

- Supervised method - Where a target variable is specified – The algorithm "learns" from the examples by determining which values of the predictor variables are associated with different values of the target variable.
- k-nearest neighbour classification for test set from training set. For each row of the test set, the k nearest training set vectors are found, and the classification is decided by majority vote, with ties broken at random. If there are ties for the kth nearest vector, all candidates are included in the vote.

KNN Parameters

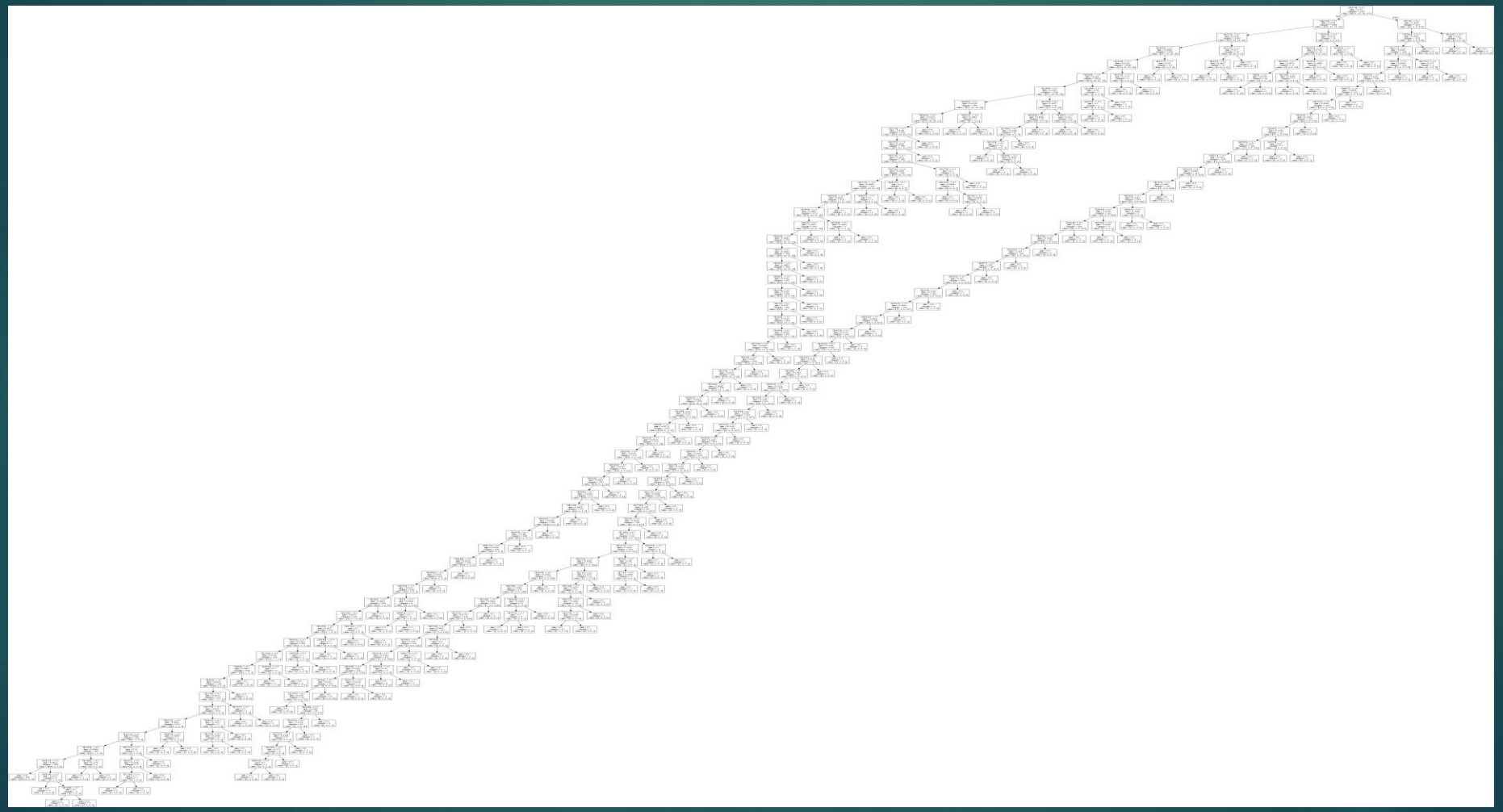

```
#=============================================================================
#build the parameter grid
KNN_grid = [{'n_neighbors': [1,3,5,7,9,11,13,15,17], 'weights':['uniform','distance']}]

#build a grid search to find the best parameters
gridsearchKNN = GridSearchCV(KNN_classifier, KNN_grid, cv=5)

#run the grid search
gridsearchKNN.fit(counts_train,labels_train)

#=============================================================================
```

Output

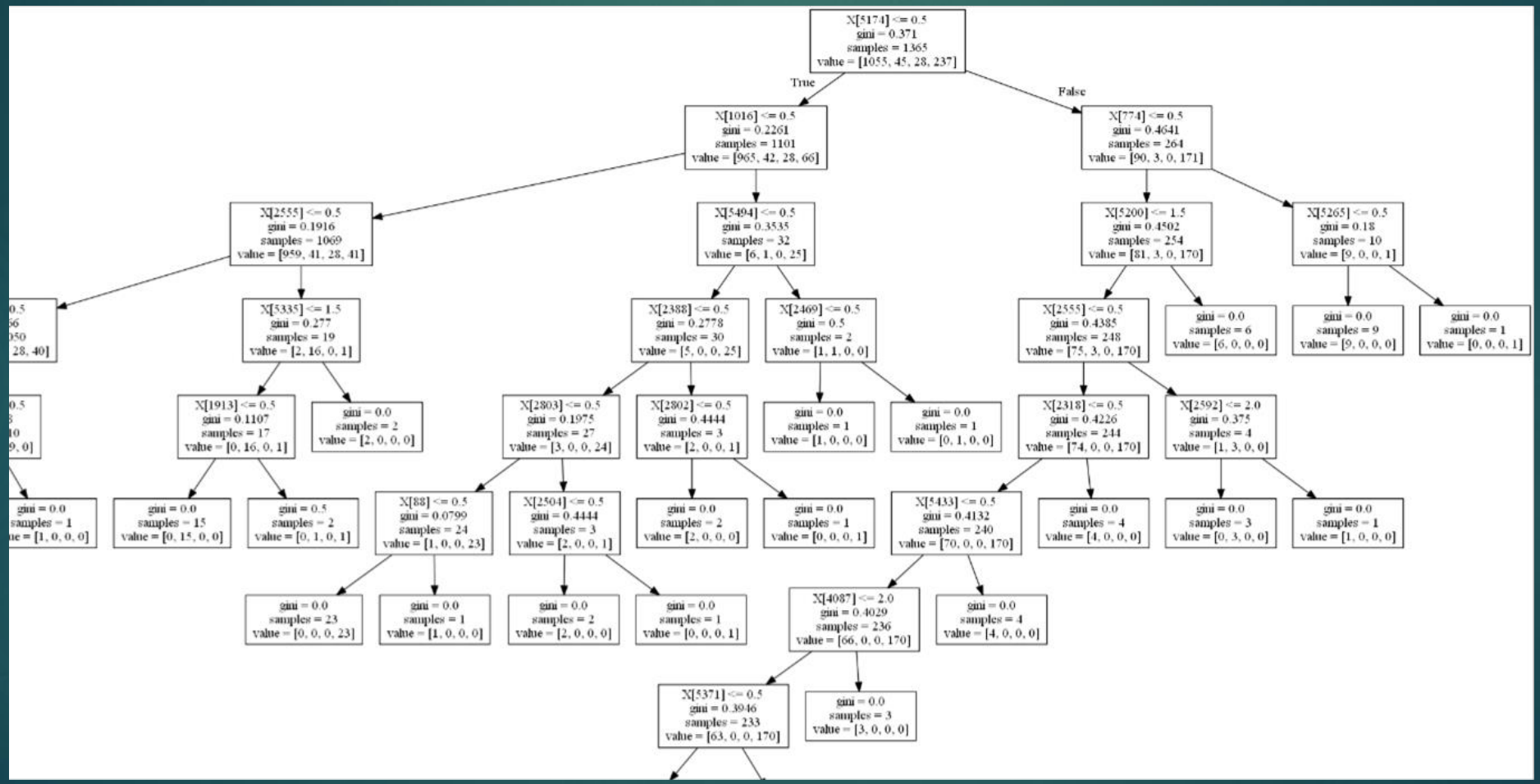
- {'n_neighbors': 1, 'weights': 'uniform'} 0.7670329670329671
- {'n_neighbors': 1, 'weights': 'distance'} 0.7670329670329671
- {'n_neighbors': 3, 'weights': 'uniform'} 0.7772893772893773
- {'n_neighbors': 3, 'weights': 'distance'} 0.7714285714285715
- {'n_neighbors': 5, 'weights': 'uniform'} 0.8131868131868132
- {'n_neighbors': 5, 'weights': 'distance'} 0.8043956043956044
- {'n_neighbors': 7, 'weights': 'uniform'} 0.8278388278388278
- {'n_neighbors': 7, 'weights': 'distance'} 0.819047619047619
- {'n_neighbors': 9, 'weights': 'uniform'} 0.8263736263736263
- {'n_neighbors': 11, 'weights': 'uniform'} 0.8293040293040294
- {'n_neighbors': 13, 'weights': 'distance'} 0.8197802197802198
- {'n_neighbors': 15, 'weights': 'uniform'} 0.8336996336996337
- {'n_neighbors': 17, 'weights': 'distance'} 0.8219780219780219

Accuracy:
0.8336996336996337

Decision Tree



Decision Tree



DT Parameters


```
#=============================================================================

#build the parameter grid
DT_grid = [{'max_depth': [3,4,5,6,7,8],'criterion':['gini','entropy']}]

#build a grid search to find the best parameters
gridsearchDT  = GridSearchCV(DT_classifier, DT_grid, cv=5)

#run the grid search
gridsearchDT.fit(counts_train,labels_train)

DT = tree.DecisionTreeClassifier()
treeClf = DT.fit(counts_train,labels_train)
tree.export_graphviz(treeClf, out_file='DTree.dot')
#=============================================================================
```

Output

- {'criterion': 'gini', 'max_depth': 3} 0.8586080586080586
- {'criterion': 'gini', 'max_depth': 4} 0.8622710622710623
- {'criterion': 'gini', 'max_depth': 5} 0.8666666666666667
- {'criterion': 'gini', 'max_depth': 6} 0.8593406593406593
- {'criterion': 'gini', 'max_depth': 7} 0.8578754578754578
- {'criterion': 'gini', 'max_depth': 8} 0.8556776556776556
- {'criterion': 'entropy', 'max_depth': 3} 0.8549450549450549
- {'criterion': 'entropy', 'max_depth': 4} 0.8630036630036630
- {'criterion': 'entropy', 'max_depth': 5} 0.8695970695970696
- {'criterion': 'entropy', 'max_depth': 6} 0.8681318681318682
- {'criterion': 'entropy', 'max_depth': 7} 0.8652014652014652
- {'criterion': 'entropy', 'max_depth': 8} 0.8593406593406593

Accuracy:
0.8695970695970696

Logistic Regression

- It is a classification method that generalizes logistic regression to multiclass problems, i.e. with more than two possible discrete outcomes
- It is used to predict the probabilities of the different possible outcomes of a categorically distributed dependent variable, given a set of independent variables

LR Parameters


```
#=============================================================================

#build the parameter grid
LREG_grid = [ {'C':[0.5,1,1.5,2],'penalty':['l1','l2']}]

#build a grid search to find the best parameters
gridsearchLREG  = GridSearchCV(LREG_classifier, LREG_grid, cv=5)

#run the grid search
gridsearchLREG.fit(counts_train,labels_train)

#=============================================================================
```

Output

- {'C': 0.5, 'penalty': 'l1'} 0.8293040293040294
- {'C': 0.5, 'penalty': 'l2'} 0.8307692307692308
- {'C': 1, 'penalty': 'l1'} 0.8315018315018315
- {'C': 1, 'penalty': 'l2'} 0.8315018315018315
- {'C': 1.5, 'penalty': 'l1'} 0.8300366300366301
- {'C': 1.5, 'penalty': 'l2'} 0.832967032967033
- {'C': 2, 'penalty': 'l1'} 0.8278388278388278
- {'C': 2, 'penalty': 'l2'} 0.8322344322344323

Accuracy:
0.832967032967033

Random Forest

- Random forest (or random forests) is an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the class's output by individual trees.
- It Operates by constructing many decision trees

RF Parameters


```
#=============================================================================
#build the parameter grid
RF_grid = [{'n_estimators':[500,1000,1500,2000,2500,3000], 'criterion':['gini','entropy']}]

#build a grid search to find the best parameters
gridsearchRF = GridSearchCV(RF_classifier, RF_grid, cv=5)

#run the grid search
gridsearchRF.fit(counts_train,labels_train)


#=============================================================================
```

Output

- {'n_estimators': 500, 'criterion': 'gini'} 0.8468864468864469
- {'n_estimators': 1000, 'criterion': 'gini'} 0.8498168498168498
- {'n_estimators': 1500, 'criterion': 'gini'} 0.8468864468864469
- {'n_estimators': 2000, 'criterion': 'gini'} 0.8461538461538461
- {'n_estimators': 2500, 'criterion': 'gini'} 0.8490842490842491
- {'n_estimators': 3000, 'criterion': 'gini'} 0.8476190476190476
- {'n_estimators': 500, 'criterion': 'entropy'} 0.8446886446886447
- {'n_estimators': 1000, 'criterion': 'entropy'} 0.8454212454212454
- {'n_estimators': 1500, 'criterion': 'entropy'} 0.8424908424908425
- {'n_estimators': 2000, 'criterion': 'entropy'} 0.8454212454212454
- {'n_estimators': 2500, 'criterion': 'entropy'} 0.8454212454212454
- {'n_estimators': 3000, 'criterion': 'entropy'} 0.8424908424908425

Accuracy:
0.8498168498168498

Naive Bayesian

- It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors.
- In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.
- A Naive Bayesian model is easy to build, with no complicated iterative parameter estimation which makes it particularly useful for very large datasets.

NB Parameters


```
#=============================================================================
#build the parameter grid
NB_grid = [{'alpha':[0,1.0,5.0,10.0,20.0,30.0,40.0], 'fit_prior':[True,False]}]

#build a grid search to find the best parameters
gridsearchNB = GridSearchCV(NB_classifier, NB_grid, cv=5)

#run the grid search
gridsearchNB.fit(counts_train,labels_train)

#=============================================================================
```

Output

{'alpha': 0, 'fit_prior': True} 0.7970695970695971
{'alpha': 0, 'fit_prior': False} 0.7919413919413919
{'alpha': 1.0, 'fit_prior': True} 0.7934065934065934
{'alpha': 1.0, 'fit_prior': False} 0.7736263736263737
{'alpha': 5.0, 'fit_prior': True} 0.802930402930403
{'alpha': 5.0, 'fit_prior': False} 0.8131868131868132
{'alpha': 10.0, 'fit_prior': True} 0.780952380952381
{'alpha': 10.0, 'fit_prior': False} 0.7992673992673993
{'alpha': 20.0, 'fit_prior': True} 0.7736263736263737
{'alpha': 20.0, 'fit_prior': False} 0.7846153846153846
{'alpha': 30.0, 'fit_prior': True} 0.7743589743589744
{'alpha': 30.0, 'fit_prior': False} 0.7846153846153846
{'alpha': 40.0, 'fit_prior': True} 0.7743589743589744
{'alpha': 40.0, 'fit_prior': False} 0.7831501831501831

Accuracy:
0.8131868131868132

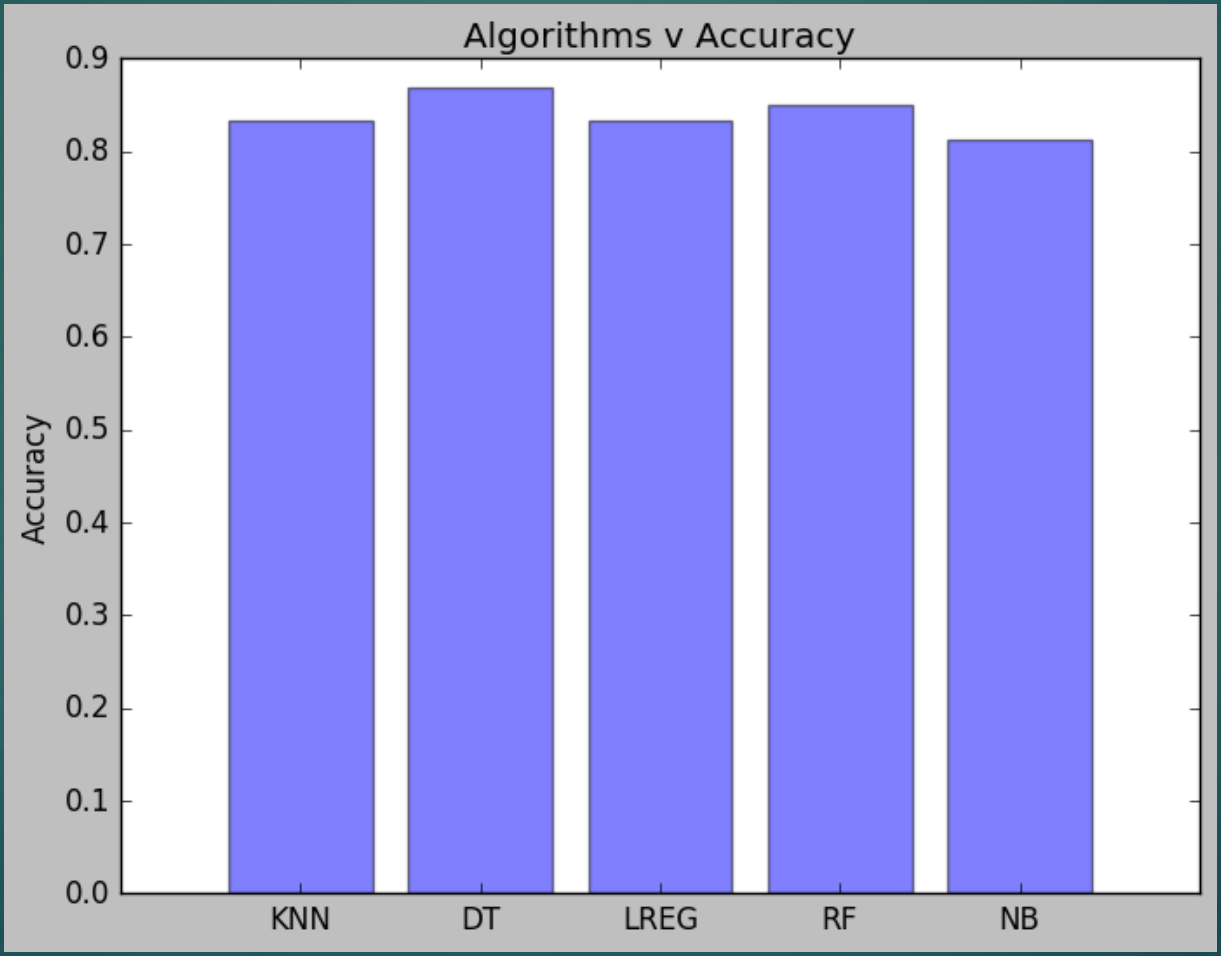
ANN - Artificial Neural Networks


```
for line in fin: # for every line in the file (1 review per line)

    line=line.lower().strip()

    dataLine=line.split('\t')
    if(len(dataLine) == 3):
        if(int(dataLine[1]) > -1):
            for i in range(int(dataLine[1]) + 1):
                tags.append(dataLine[0])
                votes.append(dataLine[1])
                questions.append(dataLine[2])
```

Highest Accuracy

Accuracy:
0.916256157635

Comparison of Classification models



Conclusion

- We have explored different prediction models. By measuring the performance of the models using real data, we have seen interesting results on the predictability of the category of questions.
- We found out that ANN has the maximum accuracy for predicting the Career Cup dataset.

Future Scope

- Currently we are using 4 categories for prediction. But in future, this can be extended to hundreds of categories with a satisfactory accuracy.
- In future, we can add a categorization for the question according to the company as well. For example, what kind of questions are asked for Amazon or Google could be predicted.

References

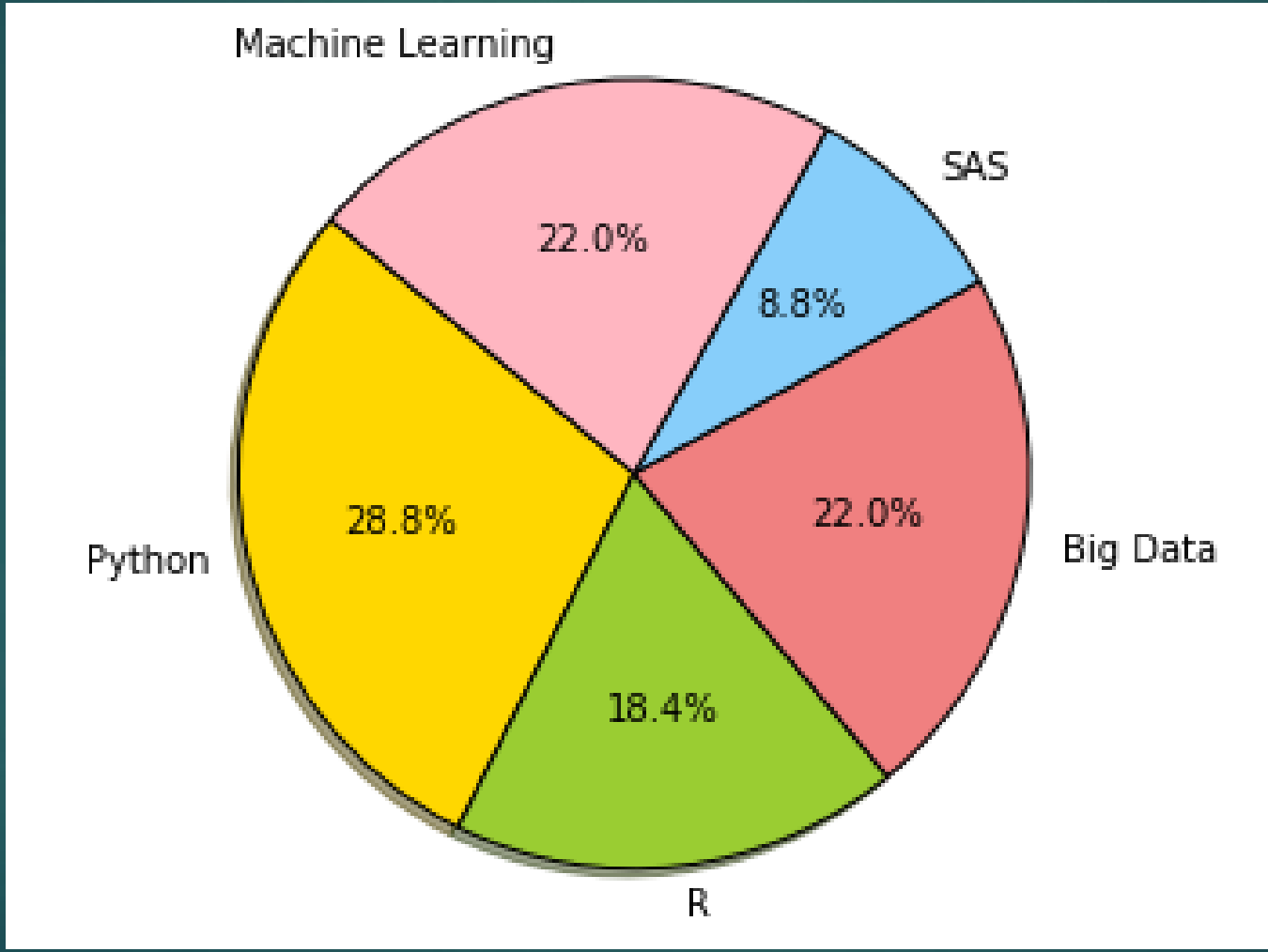
- http://www.careercup.com
- https://www.wikipedia.org/

Data Science Required Job Analysis

Data Science Job Analysis - Monster.com

- Total Jobs: **450**
  - Total Python Skill Jobs: 200
    - **Python** Percentage: **44.44%**
  - Total Big Data Skill Jobs: 153
    - **Big Data** Percentage: **34.00%**
  - Total SAS Skill Jobs: 61
    - **SAS** Percentage: **13.56%**
  - Total R Skill Jobs: 128
    - **R** Percentage: **28.44%**
  - Total Machine Learning Skill Jobs: 153
    - **Machine Learning** Percentage: **34.00%**

Result:



Thank You

Question & Answer