**A REPORT ON HOW TO IMPROVE THE WEBSITE** [**https://isthisarealjob.com/**](https://isthisarealjob.com/) **USING MACHINE LEARNING TO PREDICT IF A JOB IS REAL OR FAKE**

The problem here is similar to spam filtration and fraud detection. So it is a classification problem. We are classifying real jobs and fake jobs.

Before we can create a Machine Learning Model, we need data. So we are going to need a dataset containing both real and fake jobs. We will then use a classification algorithm to predict whether a job is real or fake.

Some of the details in the text may use includes; Name of Company, Job Description, Qualifications needed, Educational background, Email address of the company, etc

If there is no company name, then the probability of the job being a fake is high. Also if the company’s email address uses the domain, [XYZ@gmail.com](mailto:XYZ@gmail.com) or [XYZ@yahoo.com](mailto:XYZ@yahoo.com), then the probability of the job being a fake is also high. In the Result column, we use ‘Fake’ and ‘Real’ to show fake and real jobs.

**DATA COLLECTION**

Data can be collected from websites like LinkUp.com, LinkedIn.com, Indeed.com, CareerBuilder.com, Robert Half, etc. Data for real jobs can be collected from these sites. We could also get data from newspapers about job postings. Data about fake and real jobs can be gotten from Jobberman.

From my research, I found out that “the most successful AI projects are those that integrate a data collection strategy during the service life-cycle”. So every time a user uses the service, we need to collect data from the interaction with that user. So every new user makes the dataset bigger and thus the service better because more accurate models will be created.

The search query of the users could be stored in a database. So it can be added to our dataset. Also we could allow users to freely contribute, by supplying labelled data, which could be added to our dataset after authentication.

**LIBRARIES TO BE USED**

We will import the following libraries;

-Numpy -Pandas -nltk (natural language toolkit) -string

We will also need the class ‘stop words’ from the nltk.corpus module

We will use the nltk library to remove stop words from the text.

Stop words are commonly used words that should be ignored.

We use pandas to read the dataset and create a data-frame.

**HANDLING DUPLICATES AND MISSING VALUES**

So in our dataset, we need to remove duplicate entries, we can do that by using the pandas .drop\_duplicates() method.

We also need to check for the number of missing data for each column in our dataset. We use the pandas .isnull().sum() method to check for the total number of missing data. If there are missing data, we can solve the problem by removing the row entirely. Sometimes, Pandas may not be able to detect missing data, that is, when they contain texts like [‘n/a’, ‘na’, ‘-’, ‘--’]. So we can put them in a list, so when we import the data, these missing values will be recognized by Pandas.

# Making a list of missing value types  
missing\_values = ["n/a", "na", "--", "-"]  
df = pd.read\_csv("jobs.csv", na\_values = missing\_values)

**TOKENIZATION**

In the text column, we need to apply tokenization. We can create a function to process the texts. In the function, we remove punctuations and stop words. We use the string.punctuation method to remove punctuation. We separate each word into tokens by using the .split() method. We remove the English stop words from the text.

After tokenization, we convert the text to a matrix of token counts by using the CountVectorizer class from the sklearn.feature\_extraction.text library.

**SPLITTING DATASET TO TRAIN AND TEST SETS**

We can now split 80% of the data for training and the other 20% for testing. This can be easily done using the train\_test\_split function from the scikit-learn library. This will help us partion our data into a train and test data.

**CLASSIFICATION MODEL**

Since we have a classification problem, we can use the ‘Naïve Bayes Classifier’. We will use Multinomial Naïve Bayes Classifier which is suitable for classification of discrete values (‘Fake’ or ‘Real’).

The MultinomialNB class is imported from the sklearn.naive\_bayes library. We can train our model with this class by fitting the class to our training data.

**EVALUATION OF MODEL ON TRAINING DATA**

To know how accurate the model is, I’ll evaluate the model on the training dataset. From the sklearn.metrics library I’ll import classification\_report, confusion\_matrix, accuracy\_score classes. After importing these classes, I can now print the classification\_report which will show me the precision and recall. I can also print the confusion matrix. The confusion matrix is a performance measurement for machine learning classification. The precision and the recall are calculated using the confusion matrix. I can also print the accuracy score of the model on the training dataset. The highest accuracy score is 1.00, which is 100%. So the closer our accuracy is to 1, the better the model.

**EVALUATION OF MODEL ON TEST DATA**

Now I’ve seen how well the model does on the train data. I can also evaluate the model on the test data. Using the same classes mentioned above, I can check the accuracy score of the model on the test data. If the model accuracy is not too good, it’s because our dataset is not large enough, so we need to add more data to our dataset to improve our model.

**VISUALIZATION OF CONFUSION MATRIX**

Using the .heatmap() method from seaborn, we can visualize the confusion matrix.

**EXTRA FEATURE FOR THE WEBSITE**

Also I mentioned earlier, that we can help the user by recommending jobs that are similar to the search query of the user. If the prediction the user gets is ‘Fake’, then we can use recommender system to recommend a job that is real and also similar to the input the user gave. The recommender system will use word2vec to get similar jobs for the user from our database.

**SLACK NAME: @Ini**