Prediction-of-Startup-Acquisition-status-DST15072022

Website Link: https://www.crunchbase.com/

- The dataset is all about the Startup's Financial Information and we need to predict the current financial status of the company.

- The dataset is of Crunchbase - 2013, database, company, investors and all the other details belongs to the company.

- In this project, we analyzed the company's status i.e., Operating, Acquired, IPO or Closed.
- The dataset is extremely biased:

Operating	Acquired	IPO	Closed
85.6%	9.4%	1.9%	3.1%

- **Understand the dataset.**

  - The dataset is completely based on the company's information from company's website database of 2013.

  - Features: All the columns with the company's information like name, permalink, category, funding dates, funding rounds, funding amount, city, state, founding dates, last milestone dates and many more.

  - Label: Status.

- **EDA.**

  - Shape of our Dataset (196553, 44)

  ** data.shape **

  - Descriptive Statistic.

  ** data.describe() **

Unnamed:0.1entity_idparent_idlogo_widthlogo_heightinvestment_roundsinvested_companies funding_roundsfunding_total_usdmilestonesrelationshipslatlngROI

count196553.000000196553.0000000.0110110.000000110110.0000002591.0000002591.0000 0031707.0000002.787400e+0491699.000000129667.00000083852.00000083852.000000726.0 00000

mean98276.000000153006.227333NaN459.132685222.7289172.3720572.2049401.6597601.4 81652e+071.1994022.85206737.56451252.12306645.745037std56740.10806790209.250941N aN594.982577333.09072212.17351011.4369551.2016666.775937e+070.5400999.10030915.47 710270.049067572.035638 min0.0000001.000000NaN1.0000001.0000001.0000001.0000001.0000002.910000e+021.0000001.00000050.942326159.4977460.011111

25%49138.00000059850.000000NaN192.00000070.0000001.0000001.0000001.0000005.000000e+051.0000001.00000034.052234111.9400052.648879

50%98276.000000174539.000000NaN267.000000105.0000001.0000001.0000001.0000002.564500e+061.0000001.00000039.76840377.0368716.500497

75%147414.000000232655.000000NaN484.000000232.0000001.0000001.0000002.0000001.100000e+071.0000003.00000045.4215300.12775813.549900

max196552.000000286215.000000NaN18200.0000009600.000000478.000000459.00000015.0000005.700000e+099.0000001189.00000077.553604176.21254913333.333333

- Information of dataset.

** data.info() **

The dataset had a lot of null values only 9 columns out of 44 had 0 null values.

Range Index: 0 - 196552

dtypes : float (12), int (2), object (30)

- Duplicate Values.

** data.duplicated().sum() **

But there are no duplicated values present in our dataset.

- Null Values.

** data.isna().sum() **

So many null values present in our dataset, there are only 9 columns with 0 null values and some of the columns are of completely null values or empty columns, also some of the columns only have few entries.

- **Data Cleaning.**

  - Drop all those unnecessary columns with all the null values and will not require in further analysis.

  ** data.drop(columns=[]) **

Dropped columns: ['id', 'Unnamed:0.1', 'normalized_name', 'parent_id', 'domain', 'homepage_url', 'logo_url', 'logo_width', 'logo_height', 'overview', 'short_description', 'twitter_username', 'description', 'tag_list', 'country_code', 'state_code', 'city', 'first_investment_at', 'last_investment_at', 'first_milestone_at', 'last_milestone_at', 'first_funding_at', 'last_funding_at', 'lat', 'lng']

- Dropping the null rows from dataset, as they will be having unique entries only.

** data.dropna(subset=[]) **

Dropping null values: ['name', 'country_code', 'category_code', 'founded_at']

After dropping all, our dataset shape: 64094, 16

- Filling the missing values.

** data[columns_name]= data[columns_name].fillna(data[columns_name].mean()) **

** data[columns_name]= data[columns_name].fillna(data[columns_name].median()) **

** data[columns_name]= data[columns_name].fillna(data[columns_name].mode()) **

Depending upon the columns, we filled the missing values/null values in our coulmns.

- Date-Time columns.

** data[column_name]= pd.to_datetime(data[column_name]) **

Converted the datetime columns as per our use.

- Updated the ‘Closed_at’ column

```
** for i in data['status']:
   if (i == 'operating' or 'ipo'):
      data['closed_at'].fillna(2021,inplace = True)
   elif (i == 'accquired' or 'closed'):
      data['closed_at'].fillna(2012, inplace = True) **
```

~ If Status is operating or ipo, fillna with 2021

~ If Status is accquired or closed, fillna with mode() i.e. 2012.

- Adding new updated columns.

** data['companies_age'] = data['closed_at'] - data['founded_at'] **

- Reset the index.

** data.reset_index(inplace = True) **

- Shape of our cleaned dataset: 64094, 18

- **Visualization (Insights)**

  - Bivariate analysis.

  ** plt.scatterplot(x, y, data = data) **

  Plotting the scatter graph on different columns, for visualizing the relationship between two columns.

- Univariate analysis.

** sns.histplot(data[column]) **

Plotting histplot to visualize the contribution of each entry present in the column.

- **Correlation.**

  ** sns.heatmap(data.corr(), annot = **True**) **

  Plotting the heatmap to see the multicollinearity between the columns.

  - ✗ Investment Round and Invested Companies columns are highly correlated columns.
  - ✗ Founted at and Companies age columns are also highly correlated columns.

  - Pairplot.

  ** sns.pairplot(data, diag_kind = 'kde') **

- **Normal Distribution and Outliers.**

  ** sns.distplot(data[columns]) ** (Normal Distribution)

  ** sns.boxplot(data[columns]) ** (Outliers)

  After plotting the graphs and checking it we came to these conclusions:

  - ✗ Dataset is not normally distributed.
  - ✗ Dataset is very much skewed.
  - ✗ Dataset will also have many outliers.

  - Removing Outliers.

  ** IQR = Q3 - Q1 **

  ** Q3.column_name **+** (1.5 * IQR.column_name) **

  - ✓ We removed outliers from funding_rounds and funding_total_usd columns only.

  We did not remove every outlier of the rest of our dataset because:

  - ✗ Removing all outliers will lead to data leakage.
  - ✗ 2695 rows × 29 columns, is the shape of our dataset, after removing the outliers.
  - ✗ It's also not giving the best accuracy for the model.
  - ✗ For RandomForest Model the accuracy is approx. 75%
  - ✗ For KNN Model the accuracy is approx. 70%

- **Encoding.**

  ** pd.get_dummies(data, columns = ['category_code','country_code']) **

Encoding category code and country code columns by getting the top 10 repeated values using replace and get dummies method.

- After Encoding the shape of our dataset: 60069, 35

- **Imbalanced Target Column.**

  - We converted the status column to binary ‘1’ if the status was Operating or IPO and ‘0’ if Acquired or closed.

  - After checking the counts we noticed that our target column is imbalanced.

  - Balanced the target column using down-sampling approach.

  ** ds = NearMiss(0.75)
  x_train_nm,y_train_nm = ds.fit_resample(x_train,y_train) **

  - Before Fit: Counter({1: 41459, 0: 3592})
  - After Fit: Counter({1: 4789, 0: 3592})

- **PCA.**

  - We tried to reduce it to 2 feature using PCA.

  ** X = x_train_nm.copy()

  pca = PCA(n_components=2)

  pca_X= pca.fit_transform(X) **

  - But it wasn’t useful, from the graph, you can observe that there is no best_fit_line can divide the dataset into 2 classes.

  ***** FINAL DATASETE SHAPE IS 60069 ROWS × 33 COLUMNS *****

- **Scaling the Dataset.**

  - Separating the Features & Target Column.

  ** x = data.drop('status',axis = 1)

  y = y_train_nm **

  - Using Standard Scaler to scale the features.

  ** scaler = StandardScaler()

  x_scaled = scaler.fit_transform(x) **

  - fit_transform() : is used on the training data so that we can scale the training data and also learn the scaling parameters of that data. Here, the model built by us will learn the mean and variance of the features of the training set. These learned parameters are then used to scale our test data.

- transform() : uses the same mean and variance as it is calculated from our training data to transform our test data. Thus, the parameters learned by our model using the training data will help us to transform our test da ta. As we do not want to be biased with our model, but we want our test da ta to be a completely new and a surprise set for our model.

- **Model Building.**

  - Metrics considered for Model Evaluation.

  ** Accuracy , Precision , Recall and F1 Score **

  - ✓ Accuracy: What proportion of actual positives and negatives is correctly classified?
  - ✓ Precision: What proportion of predicted positives are truly positive ?
  - ✓ Recall: What proportion of actual positives is correctly classified ?
  - ✓ F1 Score : Harmonic mean of Precision and Recall.

  - Confusion Matrix.

  A confusion matrix visualizes and summarizes the performance of a classification algorithm.

  - ROC Curve.

  An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters: True Positive Rate. False Positive Rate.

➢ **RandomForest Model.**

** RandomForestClassifier() **

The random forest is a classification algorithm consisting of **many decision trees.** It uses bagging and features randomness when building each individual tree to try to create an uncorrelated forest of trees whose pre diction by committee is more accurate than that of any individual tree. - **Bagging and Boosting**: In this method of merging the same type of pre dictions. Boosting is a method of merging different types of predictions. Bagging decreases variance, not bias, and solves overfitting issues in a model. Boosting decreases bias, not variance. - **Feature Randomness**: In a normal decision tree, when it is time to split a node, we consider every possible feature and pick the one that produces the most separation between the observations in the left node vs. those in the right node. In contrast, each tree in a random forest can pick only from a random subset of features. This forces even more variation amongst the trees in the model and ultimately results in lower correlation across trees and more diversification.

- Train Test Split.

** x_train,x_test,y_train,y_test = train_test_split(x_scaled, y, test_size = 0.25,

random_state = 8) **

x_train,y_train are training data values.

x_test,y_test are testing data values.

Test size is 25%

Random State is 8

- Accuracy Score = 95.1%

- Confusion Matrix

False Positive (Type 1 Error) = 675.

False Negative (Type 2 Error) = 60.

- ROC = 88% AUC (Area Under the Curve)

- Hyperparameter Tuning.

Post tuning results are lower than the previous default parameter, so we will go with the default parameters only.

- Pipeline.

** pipeline_randomforest = Pipeline([('scaler1', StandardScaler()),

('pca1',PCA(n_components=2)),

('rf_classifier',RandomForestClassifier())]) **

** pipeline_randomforest.fit(x_train,y_train)

pred = pipeline_randomforest.predict(x_test) **

- Accuracy score=93%

- **KNearest Neighbor Model.**

  ****** KNeighborsClassifier() **

  KNearest Neighbor algorithm falls under the Supervised Learning category and is used for classification (most commonly) and regression. It is a versatile algorithm also used for imputing missing values and resampling datasets. As the name (K Nearest Neighbor) suggests it considers K Nearest Neighbors (Data points) to predict the class or continuous value for the new Datapoint. The algorithm's learning is:

1. Instance-based learning: Here we do not learn weights from training data to predict output (as in model-based algorithms) but use entire training instances to predict output for unseen data.

2. Lazy Learning: Model is not learned using training data prior and the learning process is postponed to a time when prediction is requested on the new instance.

3. Non -Parametric: In KNN, there is no predefined form of the mapping function.

For classification: A class label assigned to the majority of K Nearest Neighbors from the training dataset is considered as a predicted class for the new data point.

For regression: Mean or median of continuous values assigned to K Nearest Neighbors from training dataset is a predicted continuous value for our new data point.

From both the error curve and accuracy curve we can see that the value of error as well as accuracy becomes nearly constant for values of K equal to 7 or 8.

- KNN model - K=8

Accuracy=94%

- KNN model –K=7

Accuracy= 95%

- Hyperparameter Tuning.

** {'metric': 'minkowski', 'n_neighbors': 11, 'weights': 'uniform'} ** (Best Parameters.)

As we can conclude that, post tuning score is very close to the previous accuracy score, so we can consider that our model is giving best accuracy score in both of the cases before and after tuning.

- Pipeline.

** pipeline_KNeighborsClassifier= Pipeline([('scaler2', StandardScaler()), ('pca2',PCA(n_components=2)), ('kNN_classifier',KNeighborsClassi fier(n_neighbors = 7))]) **

** pipeline_KNeighborsClassifier.fit(x_train,y_train)

pred = pipeline_KNeighborsClassifier.predict(x_test) **

Accuracy score=93%

- **Quadratic Discriminant Analysis.**

  ****** QuadraticDiscriminantAnalysis() **

  QDA is a variant of LDA in which an individual covariance matrix is estimated for every class of observations. QDA is particularly useful if there is prior knowledge that individual classes exhibit distinct covariances. A disadvantage of QDA is that it cannot be used as a dimensionality reduction technique. In QDA, we need to estimate ΣkΣk for each class k∈{1,…,K}k∈{1,…,K} rather than assuming Σk=ΣΣk=Σ as in LDA. The discriminant function of LDA is quadratic in xx: δk(x)=−12log|Σk|−12(x−μk)TΣ−1k(x−μk)+logπk.δk(x)=−12log⁡|Σk|−12(x−μk)TΣk−1(x−μk)+log⁡πk. Since QDA estimates a covariance matrix for each class, it has a greater number of effective parameters than LDA. We can derive the number of parameters in the following way. We need KK class priors πkπk. Since ∑Ki=1πk=1∑i=1Kπk=1, we do not need a parameter for one of the priors. Thus, there are K−1K−1 free parameters for the priors. Since there are KK centroids, μkμk, with pp entries each, there are KpKp parameters relating to the means. From the covariance matrix, ΣkΣk, we only need to consider the diagonal and the upper right triangle. This region of the covariance matrix has p(p+1)2p(p+1)2 elements. Since KK such matrices need to be estimated, there are Kp(p+1)2Kp(p+1)2 parameters relating to the covariance matrices. Thus, the effective number of QDA parameters is K−1+Kp+Kp(p+1)2K−1+Kp+Kp(p+1)2. Since the number of QDA parameters is quadratic in pp, QDA should be used with care when the feature space is large.

  - Accuracy Score = 94.8%

  - Confusion Matrix

  False Positive (Type 1 Error) = 767.

  False Negative (Type 2 Error) = 15.

  - ROC = 69% AUC (Area Under the Curve)

  - Hyperparameter Tuning.

  ** {'reg_param': 1e-05, 'store_covariance': True, 'tol': 0.0001} ** (Best Parameters.)

  Accuracy score=94.9%

  As we can conclude that, post tuning score is very close to the previous accuracy score, so we can consider that our model is giving best accuracy score in both of the cases before and after tuning, So can go with both of the parameters.

  - Pipeline.

** pipeline_QDA= Pipeline([('Scaler', StandardScaler()), ('PCA',PCA(n_components=2)), ('QDA',QuadraticDiscriminantAnalysis())]) **

** pipeline_QDA.fit(x_train,y_train)

y_pred = pipeline_QDA.predict(x_test) **

Accuracy score=92%

- **Deployment.**

  ** USING DJANGO & HEROKU **

  DJANGO

  - Django is a back-end server-side web framework.
  - Django is free, open source and written in Python.
  - Django makes it easier to build web pages using Python.

  HEROKU

  - Heroku is a cloud Platform as a Service (PaaS).
  - Heroku supporting several programming languages.
  - Heroku is one of the first cloud platform.

  App Link: https://technocolabsteamb.herokuapp.com/

  GitHub Link: https://github.com/Miteshverma9/heroku

  ** USING FLASK & HEROKU **

  FLASK

  - Flask is a web application framework written in Python.
  - Flask is based on the Werkzeuge WSGI toolkit and Jinja2 template engine.

  HEROKU

  - Heroku is a cloud Platform as a Service (PaaS).
  - Heroku supporting several programming languages.
  - Heroku is one of the first cloud platform.

  App Link: https://technocolabsteambflask.herokuapp.com

  GitHub Link: https://github.com/Miteshverma9/Heroku-Flask