Domain Name Price Prediction System

1st Aekkaraj Kuplakatee
U2235955
University of East London
London, United Kingdom
u2235955@uel.ac.uk

2nd Adeyemi Farooq Opeyemi
U2228551
University of East London
London, United Kingdom
u2228551@uel.ac.uk

3rd Prabhakar Kenneth Vemagiri
U2337009
University of East London
London, United Kingdom
u2337009@uel.ac.uk

Abstract* — *Websites are possibly the most valuable tool for doing business in the digital era. All websites across the globe need a unique name to define them, which is known as a domain name. This unique identifier is used to access the website by typing it into a web browser. The valuation of a domain name can vary depending on several factors, such as its perceived value, length and the availability of similar names. The domain name price prediction system would be a tool that could help individuals and businesses determine the likely price of a particular domain name. The aim of this paper is to design and implement the domain name price prediction system. In our work, we extracted eight lexical features and three dynamic features by using the python-whois technique from domain name in a dataset and built the model with three classifications, which are K-Nearest Neighbour, Random Forest, and Decision Tree in order to compare the prediction against the category column on the dataset. The results from all classifiers - with and without parameters - delivered high levels of accuracy.

Keywords* — *Domain Name, Machine Learning, Price Prediction

I. INTRODUCTION

A domain name is a unique identifier for a website, which is used to access it by typing the name into a web browser. The price of a domain name depends on various features, such as the length of the domain name and a meaning of word in the domain name. Whilst the domain name price prediction system is a powerful tool that uses machine learning algorithms to estimate the value of the domain name, there has been a lack of studies in price valuation tools. The domain name price prediction system can be advantageous for individuals and businesses by helping them to make proper decisions about whether to purchase a particular domain name and can also be utilised as a negotiating tool for purchasers and merchants in the domain name market. The system uses machine learning algorithms to analyse data on past domain name sales, as well as other relevant factors to make predictions about the future price of a domain name. The system can also be used by domain name as a negotiating tool. For instance, a vendor could use the system to establish the price for a domain name, while a client could use it to determine a reasonable offer price. The development and implementation of a domain name price prediction system would likely require significant research and analysis. This would involve collecting and analysing data from previous domain name sales, as well as developing and testing machine learning algorithms that would be used to make predictions. The purpose of a domain name price prediction system is to estimate the valuation of a particular domain name. To create reliable predictions, the system would use a variety of data sources and machine learning algorithms.

Overall, the domain name price prediction system has the potential to be a valuable tool for firms and individuals involved in the domain name market. By providing accurate predictions about the possible price of a domain name, the system can help users make more informed decisions and facilitate more efficient negotiations. Machine learning techniques can be applied in a variety of ways to develop domain name price prediction systems. Some methods train machine learning models that can forecast the price of a given domain name based on its attributes using previous data on domain name sales. Other systems evaluate the content of websites linked to a domain name using natural language processing and other methods using this data to estimate the value of the domain name. Anyone working in any field can benefit from systems that forecast domain name prices, and they can help organisations choose more profitable domain names to buy and sell. These technologies will become even more important in the future as the domain name market develops and changes.

II. LITERATURE REVIEW

Moubayed A. et al, (2018) "DNS Typo-squatting Domain Detection: A Data Analytics & Machine Learning Based Approach"

The Typo-squatting Detection system was built to help users detect attacks on domains and prevent them from getting a DNS error. The ensemble learning classifier that is based on five traditional supervised machine learning classification algorithms was used to identify suspicious domains. This was done to improve prediction and reduce the bias and variance of the classifiers. Using the same equations, the proposed ensemble learning classifier's performance is evaluated in terms of metrics calculation. More features were extracted using the Lexical feature extraction approach e.g., Length of Domain Name • Number of Unique Characters • Number of Unique Letters • Number of Unique Numbers • Ratio of Letters to Domain Length • Ratio of Numbers to Domain Length. The labelled dataset receives a binary feature to display the domain class. In this instance, the DGA domain was represented by 1 and the valid domain by 0. Because of its high precision rate, the proposed ensemble learning classifier can detect domains that are fake or questionable. This displays how well the classifier performs in comparison to other classifiers. The ensemble learning classifier has the best accuracy, precision, and F-score as well as a high recall value.

Benlamine, K. et al. (2018) "Domain name recommendation based on Neural Network"

This system was built to identify available domain names associated with a particular subject or keyword. The elbow method technique was used to find the number of clusters(k), Two approaches were used in this system, the first approach consists in finding similarity between words of the domain name, while the second one in predicting TLD corresponding to the context of the domain name. The existing methods concerning text similarity do it through three approaches: Probabilistic-based approach, Prediction based approach and Hybrid approach which is the combination of the two previous approaches and were also used. Related data points were clustered together using the K-means clustering technique to create groupings that may be used to identify patterns and trends in the data. The results in this recommendation system were achieved by obtaining a sample from the 10 million Gandi purchase database and tested for accuracy. The accuracy was determined by how well the expected TLDs matched the top 10 TLDs in terms of score, and as part of the second technique, a sample of 10,000 purchase domain names were collected to set the neural network's parameters in a way that would produce the best results. Any neural network's output end can have the activation function added as a node. It is used to determine the neural network's output, such as yes or no. The obtained values are mapped between 0 and 1 or -1 and 1.

Upadhyay, S. and Ghorbani, A. (2020) "Feature extraction approach to unearth domain generating algorithms (DGAS)"

With the intention of resolving the issue of manually creating domain names, a technology known as algorithmically generated domain names (AGD) was created. Lexical feature, Web Scrapped Features and Linguistic Features extraction modules were used to extract features from the dataset, whilst examples of features extracted included the length of the domain, Subdomains plus TLDs, number to alphabet ratio, special characters, hash, number of dots. The random forest model was used to evaluate the sampled dataset, and the results showed that accuracy rose as the dataset's quantity of records rose. As soon as the sample was tested, 20% of the test data was just DGAs, and during the testing of the 20K sample set, DGAs made up most of the test data. The methodology used here was to bring the best features which supported the framework to never fail in detection and prediction by results. The proposed a technique was an amalgam of linguistic and web extracted features which consist of a feature set never considered before for DGA Detection purposes.

Yang, C. et al. (2022) "N-Trans: Parallel detection algorithm for DGA domain names,"

In this paper, they proposed to improve the malicious domain names detection system by using a novel parallel technique called N-Trans that is based on the N-gram algorithm in order to increase the prevention, detection and restrict access to a certain network or website. Accurate detection of hostile domains is essential for effective cyberattack defence. To further enhance this, a novel parallel detection model, N-Trans, based on the Transformer model and the N-gram approach was used for this system. The Transformer model was used to detect malicious domain names, boosting the model's accuracy of identification and deep learning of features including location data and text features. The experimental results show that the parallel detection model based on N-gram and the Transformer model can identify risky domain names in DGA with 96.97% accuracy. It can successfully and reliably identify malicious domain names and outperform popular harmful domain name detection algorithms.

Saleem Raja, A., Vinodini, R. and Kavitha, A. (2021) "Lexical features based malicious URL detection using machine learning techniques"

This method looks for patterns and traits that are frequently connected to malicious URLs by examining lexical features, which are the words, numbers, and symbols that make up a URL. Dynamic Features and Lexical Features are both used in this detection algorithm to extract features such as URL length, Digits count, Special Characters, Hyphen Count, Double Slash Count, Single Slash Count, @ sign count, Protocols Count etc. Twenty percent of the data, or 2000 URLs, were utilised for testing, while the remaining eighty percent, or 8000 URLs, were used for training. Our machine learning model was trained using the machine learning classification techniques Decision Trees, KNN, Kernel SVM, Logistic Regression, Naive Bayes, Random Forest, and Support Vector Machine. The table unequivocally demonstrates that Random Forest, Decision Tree, and KNN are the top three classifiers out of all those evaluated. Random Forest, which had an accuracy rating of 87.85%, had the best results. It only misclassified 243 out of the 2000 URLs that we had predicted.

Lindenthal, T. (2014) "The price dynamics of internet domain names"

Internet websites can be found and accessed using domain names, which are normally bought through a domain name registrar. The popularity of a domain name, the desire for comparable domain names, and the market's overall supply and demand for domain names can all have an impact on how much a domain name costs. The TLD algorithm was deployed to extract features from the data in a second step, the 12 annual indices are converted to one index at monthly update frequency. All 13 indices are displayed, the fundamental or market risk of domains can be used to compare the risk of an investor's portfolio, which puts any return on this portfolio into a risk-adjusted perspective. In addition, the risk of other asset classes, such as stocks, bonds, or real estate, can be contrasted with the core risk of domains.

III. Implementation

1. Dataset

The dataset came with two columns (Figure 1), the first column labelled domain name and the second labeled category. The category column with 1 – 5 values.

	Domain	category
0	noriomatsumoto.com	1
1	haplessmales.com	1
2	shivamchatuvedi.com	1
3	dailyfreebieoffer.com	1
4	poniemall.com	1
...	...	...
9995	citymark.com	5
9996	offroading.com	5
9997	cevon.com	5
9998	mozzilla.com	5
9999	stemz.com	5

[10000 rows x 2 columns]

Figure 1 – dataset

2. Data Processing

Data Cleaning

The provided dataset was carefully checked for all missing values. Fortunately, both columns were filled with values. So, this meant the data cleaning process could be skipped.

Feature Extraction

Feature extraction is especially important for this task because it enables functionality to estimate the price for domain names. Features that were identified as useful for this task were divided into two categories, the first was lexical features and the second was dynamic features.

Lexical Features: In the dataset, the domain names have certain hidden information (Figure 2). For instance, the length of domains, special characters, and numbers. To extract the information out of the domain names, the lambda function as an argument with a loop function had been used to extract the information from the domain names.

```
# Extrat lens of the domain
DF['Length'] = DF['Domain'].astype(str).str.len()

# Extract special characters from the domians
DF["Special_chars"] = DF.apply(lambda p: sum(not q.isalpha() and not q.isdigit() for q in p["Domain"]), axis=1)

# Extrct letters from the domains including Uppercase and Lowercase
DF["Letters"] = DF.apply(lambda p: sum(q.isalpha() for q in p["Domain"]), axis=1)
DF['Upper_case'] = DF.apply(lambda p: sum(q.isupper() for q in p['Domain']), axis=1)
DF['Lower_case'] = DF.apply(lambda p: sum(q.islower() for q in p['Domain']), axis=1)

# Separate the letters to vowels and consonants
DF['Vowels'] = DF.Domain.str.lower().str.count(r'[aeiou]')
DF['Consonants'] = DF.Domain.str.lower().str.count(r'[a-z]') - DF['Vowels']

# Extract numbers from the domains
DF["Numbers"] = DF.apply(lambda p: sum(q.isdigit() for q in p["Domain"]), axis=1)
```

Figure 2 - Lexical Features

Dynamic Features: To extract the dynamic features, the whois function was used which was executed though a python-whois model. This functionality will retrieve the domain name information such as a domain name registrar and IP addresses from the severs. In this case (Figure 3) it was used to look up the register date, expiration date and the last date updated by defining it as functions.

```
# Create a function to extract the register date from the domains.
def check_registered_date(re):
    if(re!=False):
        RD = re.creation_date
        if((RD is not None) and (type(RD)!=str)):
            if(type(RD)==list):
                RD=RD[0]
            TD = datetime.datetime.now()
            days = (TD-RD).days
            return days
        else:
            return 0
    else:
        return 0
```

```
# Create the fucntion to extract the expire date from the domains.
def check_expiration_date(re):
    if(re!=False):
        ED = re.expiration_date
        if((ED is not None) and (type(ED)!=str)):
            if(type(ED)==list):
                ED = ED[0]
            TD = datetime.datetime.now()
            days = (TD-ED).days
            return days
        else:
            return 0
    else:
        return 0
```

```
# Create the function to extrct the update date from the domains.
def check_updated_date(re):
    if(re!=False):
        UDD = re.updated_date
        if((UDD is not None) and (type(UDD)!=str)):
            if(type(UDD)==list):
                UDD = UDD[0]
            TD = datetime.datetime.now()
            days = (TD-UDD).days
            return days
        else:
            return 0
    else:
        return 0
```

Figure 3 - Dynamic Features

All the features that had been extracted can affect the price for the domain names. For example, if the domains names have more than ten characters, it is likely to be cheaper. Also, domains that contain more than three numbers are likely to be cheaper than a domain with only letters. The registration date can also be tracked back to see if domains are benign. So, the older the domains the more likely to be benign which is not good for the clients wanting to purchase that domain.

3. Data Transformation

Data Encoding

In the encoding process, one hot coding was used for converting a categorical valuable that contains strings to numeric form (Figure 4). This made it readable so the machine can execute the column. In this work, the encoder gave each domain name a binary value of 0 or 1 based on which column the row's category belongs to.

```
# Use one-hot encoding to covert string column to numeric
domian_encoder = OneHotEncoder()
domian_encoder = domian_encoder.fit_transform(DF.Domain.values.reshape(-1,1)).toarray()
OneHot = pd.DataFrame(domian_encoder, columns = ["Domain_"+str(int(i)) for i in range(domian_encoder.shape[1])])

# Add the string tranformed to dataset
DF = pd.concat([DF, OneHot], axis=1)
DF = DF.drop('Domain', axis=1)
DF.head(100)
```

Figure 4 - Encoding

Feature Selection

Before the feature selection was performed, the dataset had to be divided into independent features as ‘x’, and a dependent variable as ‘y’ (figure 5)

```
# Spllt the data to X and Y
x = DF.drop('category', axis=1)
y = DF['category']
```

Figure 5 - Split X and Y

The Feature Selection technique lowers the number of variable inputs by eliminating irrelevant features from the dataset. It focuses on the most relevant features to generate the higher results for the models.

The 'Generic Univariate Select' was applied to this task as a feature selection, because it allows users to select the best features based on the univariate statistical tests with the hyper-parameter. The parameter that was chosen in this feature selection was '*fpr*' (Figure 6) which stands for 'false positive rate test' and was aimed to control the total quantity of false detections.

```
# Perform feature selection.
transform = GenericUnivariateSelect(mode="fpr")
gus = transform.fit_transform(x,y)
```

Figure 6 - Feature Selection

Data Splitting

The 'train-test split' function was used to split the dataset into training and testing. So, the training dataset is used for the features and target values that will be used when training the model. The testing dataset is used for features and target value that utilise the final evaluation of the model. The test size was spilt between the testing and training dataset. In this case, (figure 7) the dataset had been split into 30% for test and 70% for train.

```
# Spilt X and Y to train and test the dataset.
x_train, x_test, y_train, y_test = train_test_split(gus, y, test_size = 0.3, stratify=y, random_state=0)
```

Figure 7 - Train and Test dataset

Normalisation

If the dataset has not been normalised, all features will have different scales. This might lead to a biased outcome for the model, in terms of accuracy and misclassifications. So, for the finishing data process stage, the standard scaler (Figure 8) was used to normalise the dataset. Standardised features were used to remove the mean value and scale into the same unit variance.

```
# Tranform the dataset to a common standad scale by using the Standard Scaler
standardscaler = StandardScaler()
x_train = standardscaler.fit_transform(x_train)
x_test = standardscaler.transform(x_test)
```

Figure 8 - Normalisation

4. Data Modelling

In this stage, three supervision learning algorithms for classification had been applied for the domain name price prediction system.

K-Nearest Neighbour

The K-Nearest Neighbour also known as KNN or K-NN is a non-parametric type of supervision machine learning algorithm which is widely used for classification; however, it can also be used for regression problems. KNN is used to find the similarity to the underlying data by calculating the distance between the test dataset and the training point.

1.6.7.3. Mathematical formulation

The goal of NCA is to learn an optimal linear transformation matrix of size `(n_components, n_features)`, which maximises the sum over all samples i of the probability p_i that i is correctly classified, i.e.:

$$\arg\max_{L} \sum_{i=0}^{N-1} p_i$$

with N = `n_samples` and p_i the probability of sample i being correctly classified according to a stochastic nearest neighbors rule in the learned embedded space:

$$p_i = \sum_{j \in C_i} p_{ij}$$

where C_i is the set of points in the same class as sample i, and p_{ij} is the softmax over Euclidean distances in the embedded space:

$$p_{ij} = \frac{\exp(-||Lx_i - Lx_j||^2)}{\sum_{k \neq i} \exp(-||Lx_i - Lx_k||^2)}, \quad p_{ii} = 0$$

Mahalanobis distance

NCA can be seen as learning a (squared) Mahalanobis distance metric:

$$||L(x_i - x_j)||^2 = (x_i - x_j)^T M (x_i - x_j),$$

where $M = L^T L$ is a symmetric positive semi-definite matrix of size `(n_features, n_features)`.

Figure 9 - Mathematical formulation for KNN

KNN has 3 different calculations to determine the distance metrics:

- Euclidean distance: It calculates a straight line between query point (x) and an existing point (y).

$$\boldsymbol{d(x,y) = \sqrt{\sum_{i=1}^{n} (yi - xi)^2}}$$

- Manhattan distance: It calculates the absolute value between real vectors.

$$\boldsymbol{d(x,y) = (\sum_{i=1}^{n} |xi - yi|)}$$

- Minkowski distance: It is used for categorical variables. It is also referred to as the overlap metric.

$$\boldsymbol{(\sum_{i=1}^{n} |xi - yi|)^{1/p}}$$

The reason that K-Nearest Neighbour was chosen as the one of the algorithms and used for the domain name price prediction was because it is the easiest machine learning algorithm to understand. K-NN is sensitive to outliers, including complex features. It is also exclusive and offers an excellent rate of cross-validation which is useful to encounter the output.

Random Forest

Random forests are a type of machine learning algorithm that can be used to predict the class label of an input sample based on feature values. (Breiman, 2001). Random forest or RF is a supervision learning tree-based algorithm that uses ensemble learning techniques to build the model by averaging the data points randomly from several decision trees and calculating the results. It is popular for both classification and regression.

- **Regression**: residual sum of squares

  $RSS = \sum_{left} (y_i - y_L^*)^2 + \sum_{right} (y_i - y_R^*)^2$

  where y_L^* = mean y-value for left node

  y_R^* = mean y-value for right node

- **Classification**: Gini criterion

  $Gini = n_L \sum_{k=1,...,K} p_{kL} (1- p_{kL}) + n_R \sum_{k=1,...,K} p_{kR} (1- p_{kR})$

  where p_{kL} = proportion of class k in left node

  p_{kR} = proportion of class k in right node

Figure 10 - Mathematical formulation for Random Forest

Random Forest classifier was chosen because it works well with non-linear datasets and offers better final outputs then other classifications. This algorithm is suitable for use with the domain name prediction model as it can handle several variable inputs with a lower risk of getting overfitting rates.

Decision Tree

The Decision tree is a non-parametric supervision learning machine algorithm as well as KNN and it is used as a tree structure to solve classification or regression tasks. The dataset has been separated into small units of subsets called nodes while making its decision. The node divided it into various possible outcomes. Individual outcomes led to other nodes, which diversify into other probabilities. This gives it a treelike form. This algorithm uses the information gain or Gini impurity to split the data at each node. (EDUCBA 2022)

1.10.7. Mathematical formulation

Given training vectors $x_i \in R^n$, i=1,..., l and a label vector $y \in R^l$, a decision tree recursively partitions the feature space such that the samples with the same labels or similar target values are grouped together.

Let the data at node m be represented by Q_m with n_m samples. For each candidate split $\theta = (j, t_m)$ consisting of a feature j and threshold t_m, partition the data into $Q_m^{left}(\theta)$ and $Q_m^{right}(\theta)$ subsets

$$Q_m^{left}(\theta) = \{(x, y) | x_j \leq t_m\}$$
$$Q_m^{right}(\theta) = Q_m \setminus Q_m^{left}(\theta)$$

The quality of a candidate split of node m is then computed using an impurity function or loss function $H()$, the choice of which depends on the task being solved (classification or regression)

$$G(Q_m, \theta) = \frac{n_m^{left}}{n_m} H(Q_m^{left}(\theta)) + \frac{n_m^{right}}{n_m} H(Q_m^{right}(\theta))$$

Select the parameters that minimises the impurity

$$\theta^* = \operatorname{argmin}_\theta G(Q_m, \theta)$$

Recurse for subsets $Q_m^{left}(\theta^*)$ and $Q_m^{right}(\theta^*)$ until the maximum allowable depth is reached, $n_m < \min_{samples}$ or $n_m = 1$.

1.10.7.1. Classification criteria

If a target is a classification outcome taking on values 0,1,...,K-1, for node m, let

$$p_{mk} = \frac{1}{n_m} \sum_{y \in Q_m} I(y = k)$$

be the proportion of class k observations in node m. If m is a terminal node, `predict_proba` for this region is set to p_{mk}. Common measures of impurity are the following.

Gini:

$$H(Q_m) = \sum_k p_{mk}(1 - p_{mk})$$

Log Loss or Entropy:

$$H(Q_m) = -\sum_k p_{mk} \log(p_{mk})$$

Figure 11 - Mathematical formulation for Decision Tree

The decision tree was selected because it was easy to understand and utilise. Decision trees are also comprehensive which helps to consider all the possibility outcomes for the prediction. This algorithm is suitable for the domain name prediction system as it can handle both numerical and categorical data at the same time the domain name price prediction has several numerical and categorical inputs.

5. Results

Cross Validation

Cross validation is one of the most well-liked methods to test the effectiveness and evaluate performance of classification models. It is mainly used to prevent an overfitting situation. Cross validation can also be used to avoid high test error rates in the prediction.

Generally, procedures of K-Fold Cross Validation are:

1. Randomly divide the dataset into k equal parts.
2. For each k group choose k-1 folds as the training set and the rest of the data as the testing set.
3. Fit a model with a training set and validate with the testing set.
4. Repeat k time to get the accuracy by score average of the k recorded results.

In the domain name price prediction system, K-Fold was used n_split at 15. And the outcome was high which meant the inputs that had been used were relevant and unlikely to face an overfitting situation (Figure 12).

```
KNN: 0.759004 (0.008914)
Random Forest: 0.775856 (0.018080)
Decision Tree: 0.729006 (0.016139)
```

Figure 12 - K-Fold cross validation with n_split = 15

The image below (Figure 13) illustrates the accuracy of each algorithm. This can be interpreted that the Random Forest had the highest outcome and is the best model to use for the domain name price prediction system.

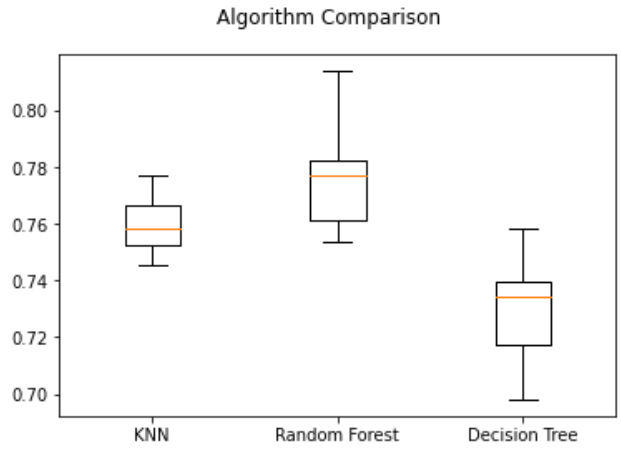


Figure 13 - Boxplot of cross validation

Therefore, this calculation was predicted using the default parameter. However, the accuracy of the results can be increased by tuning the right parameters.

Tuning Parameters

To receive the maximum model performance outcome, tuning the right parameter is an essential procedure which controls the behaviour of machine learning. If the algorithm does not tune to the right parameter, this can lead to a high error rate and lower the accuracy of the model.

The tuned parameter was used in the price prediction model to increase maximum accuracy for the results. In this case, each classification was tuned with different parameters.

- **KNN**

The k-value was used as a parameter to tune the algorithm. The best way to find the most suitable k-value for this classifier was to find the k-value with the lowest error rate (Figure 14).

```
def k_value_error():
  error_rate=[]
  for i in range (1,31):
    clf=KNeighborsClassifier(n_neighbors=i)
    clf.fit(x_train,y_train)
    predict_i=clf.predict(x_test)
    error_rate.append(np.mean(predict_i!=y_test))
    error_rate
    k_value = error_rate.index(min(error_rate)) + 1
```

Figure 14 - Find K-values

The graph below (Figure 15) presents the k-value against the error rate. K at 28 is the best value because it has the lowest error. This was used as the k-value parameter for KNN.

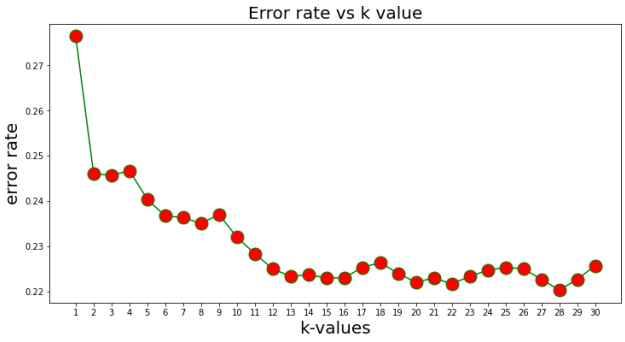


Figure 15 - plot for Error rate vs k-value

The following chart looks at the predictive model for the K-Nearest Neighbour classifier (Figure 16) it shows the accuracy at 77. 96%. The weighted average result for the precision which predicted the overall true positive prediction is 73%. And the sensitivity (recall) that's been outlined as the true positive rate is 78% which means this model has a high rate of predicting the model correctly. The weighted average balance of precision and recall is 74%.

```
K-Nearset Neighbors: 77.96666666666667 %
K-Nearset Neighbors:
 [[1784   16    0    0    0]
 [  75  487   22    4   12]
 [  37  223   22    7   11]
 [   7  114   13    2   14]
 [  10   74   12   10   44]]
K-Nearset Neighbors:
              precision    recall  f1-score   support

           1       0.93      0.99      0.96      1800
           2       0.53      0.81      0.64       600
           3       0.32      0.07      0.12       300
           4       0.09      0.01      0.02       150
           5       0.54      0.29      0.38       150

    accuracy                           0.78      3000
   macro avg       0.48      0.44      0.43      3000
weighted avg       0.73      0.78      0.74      3000
 %
```

Figure 16 - Result of KNN

- **Random Forest**

This classifier used the max depth value as a parameter to tune the algorithm. To find the best max depth value the below function was used to search for the value with the lowest error rate (figure 17).

```
def m_max_depth_for_RF():
  error_rate=[]
  for m in range (1,31):
    rfm=RandomForestClassifier(max_depth=m)
    rfm.fit(x_train,y_train)
    predict_i=rfm.predict(x_test)
    error_rate.append(np.mean(predict_i!=y_test))
    error_rate
    depth = error_rate.index(min(error_rate)) + 1
```

Figure 17 - Find max-depth for RF

The graph below shows the max-depth value against the error rate. Looking at the chart (Figure 18), the max-depth at 11 will be the best one to use as a parameter for Random Forest.

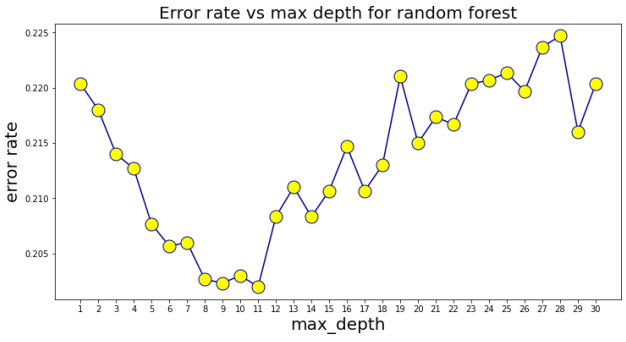


Figure 18 - chart for error rate vs max-depth for RF

The following chart looks at the predictive model for the K-Nearest Neighbour classifier (Figure 19) it shows the accuracy at 79. 53%. The weighted average result for the precision which predicted the overall true positive prediction is 75%. And the sensitivity (recall) that's been outlined as the true positive rate is 80% which means this

model has a high rate of predicting the model correctly. The weighted average balance of precision and recall is 75%.

```
Random Forest: 79.53333333333333 %
Random Forest:
 [[1782   16    2    0    0]
 [  34  538   11    3   14]
 [  12  255   11    7   15]
 [   3  111    9    7   20]
 [   4   77   13    8   48]]
Random Forest:
              precision    recall  f1-score   support

           1       0.97      0.99      0.98      1800
           2       0.54      0.90      0.67       600
           3       0.24      0.04      0.06       300
           4       0.28      0.05      0.08       150
           5       0.49      0.32      0.39       150

    accuracy                           0.80      3000
   macro avg       0.50      0.46      0.44      3000
weighted avg       0.75      0.80      0.75      3000
 %
```

Figure 19 - result of Random Forest

- **Decision Tree**

This classifier used max depth value as a parameter to tune the algorithm. To find the best max depth value as a parameter, the function below was used to look for the depth max value with the lowest error rate (figure 20).

```
def m_max_depth_for_DT():
  error_rate=[]
  for m in range (1,31):
    dtm=DecisionTreeClassifier(max_depth=m)
    dtm.fit(x_train,y_train)
    predict_i=dtm.predict(x_test)
    error_rate.append(np.mean(predict_i!=y_test))
    error_rate
    depth = error_rate.index(min(error_rate)) + 1
```

Figure 20 - Find depth-max for DT

The chart below shows the max-depth value against the error rate. Looking at the chart (Figure 21), the max-depth at 5 will be the best one to use as a parameter for the Decision Tree.

Figure 21 - graph for error rate vs depth-max for DT

The following chart looks at the predictive model for the K-Nearest Neighbour classifier (Figure 22) it shows the accuracy at 79. 4%. The weighted average result for the precision which predicted the overall true positive prediction is 76%. And the sensitivity (recall) that's been outlined as the true positive rate is 79% which means this model has a high rate of predicting the model correctly. The weighted average balance of precision and recall is 75%.

```
Decision Tree: 79.4 %
Decision Tree:
 [[1782   18    0    0    0]
 [  32  531   11    8   18]
 [   9  261    6    7   17]
 [   3  113    2    8   24]
 [   4   79    0   12   55]]
Decision Tree:
              precision    recall  f1-score   support

           1       0.97      0.99      0.98      1800
           2       0.53      0.89      0.66       600
           3       0.32      0.02      0.04       300
           4       0.23      0.05      0.09       150
           5       0.48      0.37      0.42       150

    accuracy                           0.79      3000
   macro avg       0.51      0.46      0.44      3000
weighted avg       0.76      0.79      0.75      3000
 %
```

Figure 22 - Result of Decision Tree

IV. DICUSSION

AEKKARAJ KUPLAKATEE

To accomplish this task, we first identified the dataset and the objectives we wanted to achieve. In this work, we focused on developing a domain name price system. The provided dataset contained the historical data of various domains. The first step consisted in extracting the data from the dataset in order to produce a model. The approach employed was lexical feature extraction, a method used to extract word length. Some of the domain names included both letters and numbers, so we first had to separate and count them. This meant counting vowels and consonants, as well uppercase and lowercase characters in the domain names. Then we used the py-whois model to retrieve WHOIS data, such as the domain's registration date, expiration date and the last update date. In total we extracted eleven features.

After we obtained all the required information, we next had to process the date. So first we converted the domain column from string to numeric. In this method we used the 'one hot coding' feature to transform the domain column into categorical columns in order to be able to process the entire dataset. We then split the feature values as 'X' and target values as 'Y'. For the next step, we employed the 'generic univariate selects' option for feature selection to be used in the model. Then we split train and test datasets and normalised the features with the standard scaler to set our dataset to the same scale.

In the modelling stage, we ran three different classifiers: K-Nearest neighbour, Random Forest, and

Decision tree. The accuracy of the modelling with and without tuning parameters was also tested with validation. The Random Forest (RF) classifier gave us the highest predicted accuracy, leading to the conclusion that it would be the best option for the domain price prediction system for Smart Predict. However, the Random Forest classifier could be a difficult dataset to interpret and too computationally intensive for larger datasets.

The lesson I have learned from this project is that as more features are extracted more accurate models can be produced. However, not every feature is always useful, as some might not be relevant to the dataset even if they are easy to extract. Those irrelevant features can lead the model to generate less accurate predictions. For further research in the future, I intend to extract more features and employ different prediction algorithms.

ADEYEMI FAROOQ

The approach taken to solve this problem is Identify and gather a sizable list of existing domain names, together with any pertinent details such as the categories they fall under and understanding the dataset, then pointing out the important variables that would help in predicting the domain name prediction system. Then the whois function was used for features extraction, Whois feature extraction is the process of locating and obtaining key features from the whois information of already registered domain names which is stored in databases and accessible through the whois protocol. The features extracted from the whois module are Length, Special characters, Letters, Uppercase, Lowercase, Vowels, Consonants, Numbers, registration Date, expiry Date, updated Date of the Domain Name. These new features extracted would allow us to have enough columns for our model prediction.

During the Data Pre-processing, we performed string column to numeric conversion, we then added the transformed string to the dataset, split the data into x and y then performed feature selection using generic univariate selector with mode set to ‘fpr’, then set the dataset into a standard feature because some of the variables are measured at different scales which does not make them contribute equally to the model fitting. These would allow us to have a good model fitting for our prediction.

Three different classifier algorithms were used throughout the data modelling stage: () using the k-fold and cross validation approach for measuring each algorithm's performance on the dataset, to design and evaluate the prediction models. To draw a conclusion regarding the model's prediction's accuracy, The best among the three-classifiers used is the Random Forest algorithm which gave the highest accuracy giving among the three algorithms used in this system, afterwards, the graphical representation was presented to show the Algorithm comparison between the three classifiers algorithm based on the cross validation and k-fold technique. Using a loop function, we calculated and visualised the lowest error and maximum depth to check the value for k for each classifier algorithm. To develop a prediction, the three algorithms were defined using parameters. To make it simple to compare to other classifications as well. Defined the functions to extract every result from the classification functions, and then we used a graphical representation to display the results of the classification with parameters. The optimal prediction system for domain users would therefore be the usage of the random forest algorithm.

The Computational Complexity of Random Forest algorithm depends on the number of trees in the forest, the depth of each tree, and the quantity of characteristics in the model. A random forest algorithm is described as having a complexity of O (n * m * log(m)), where n is the number of training samples and m is the number of features. However, of the three classifications utilised in this model, the random forest method is the fastest when compared to other prediction algorithms. The Decision tree, a different alternative method that we evaluated, had an accuracy rate of 79.4, This is combined to predict the target variable, reduce model overfitting, and can also handle correlations between complex features. However, when compared to the random forest classifier, we can see that the decision tree yielded lesser accuracies based on the attributes of the dataset.

The lesson that I have learned is that the greater the data we have the better models that can be made. The accuracy of the model can be predicted better in extent by how relevant the features are. In the future, I would love to apply several predictive algorithms to examine various models that can precisely represent predictions with more data and a wider range of domain names. This will make it possible to predict domain names using more sensitive and precise data.

PRABHAKAR KENNETH VEMAGIRI

The initial stage in this course work is to extract the features from the given dataset that would enable us to forecast the domain name system. In order to extract the length, special characters, letters, upper case, lower case, vowels, consonants, and numbers, we first used a simple anonymous function named lambda. Additionally, we utilised the python whois function for this as the whois protocol allows access to a particular domain name data that has already been stored in a database. Therefore, utilising these extracted characteristics, several classification algorithms have been implemented.

Before we started the classification algorithms, we performed data pre-processing which can improve the accuracy and the quality of the dataset. Because some of the variables are recorded at various scales, which prevents them from contributing equally to the model fitting, we did string column to numeric conversion, added the transformed string to the dataset, separated the data into X and Y, then performed feature selection using generic univariate selector with mode set to "fpr," and finally put the dataset into a standard feature. These would enable us to have a model that fits our forecast well for the prediction model.

As you can see from this course project, we employed three distinct classifier approaches. K-nearest Neighbour, Random Forest, and Decision Tree are

available. Additionally, we employed both the k-fold and cross validation approaches to evaluate each subsequent classifier.

In our testing, we discovered that the Random Forest Classifier enabled us to determine the model's maximum accuracy. We can also determine the minimal accuracy for the K-NN method, which gives us similar accuracy to the random forest's accuracy from the decision tree. Below is a graphical depiction that compares the three classifier algorithms based on cross validation and the k-fold method. So that, we can say. The alternative to the random forest would be the decision tree algorithm with a negligible difference in the accuracy.

And last, based on what I have learned, feature extraction can help the model be perfectly accurate even when there are not enough features to do so. By using the proper technique to the dataset, we can extract as many characteristics as we desire. As an illustration, we generated several routines to extract the characteristics from the Python whois protocol. In the future, I would like to apply feature extraction techniques to a larger variety of datasets that can improve the model's accuracy.

References:

- *Moubayed, A. et al. (2018) "DNS Typo-squatting Domain Detection: A Data Analytics & Machine Learning Based Approach," 2018 IEEE Global Communications Conference (GLOBECOM) [Preprint]. Available at: https://doi.org/10.1109/glocom.2018.8647679.*
- *Benlamine, K. et al. (2018) "Domain name recommendation based on Neural Network," Procedia Computer Science, 144, pp. 60–70. Available at: https://doi.org/10.1016/j.procs.2018.10.505.*
- *Upadhyay, S. and Ghorbani, A. (2020) "Feature extraction approach to unearth domain generating algorithms (dgas)," 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech) [Preprint]. Available at: https://doi.org/10.1109/dasc-picom-cbdcom-cyberscitech49142.2020.00077.*
- *Yang, C. et al. (2022) "N-Trans: Parallel detection algorithm for DGA domain names," Future Internet, 14(7), p. 209. Available at: https://doi.org/10.3390/fi14070209.*
- *Saleem Raja, A., Vinodini, R. and Kavitha, A. (2021) "Lexical features based malicious URL detection using machine learning techniques," Materials Today: Proceedings, 47, pp. 163–166. Available at: https://doi.org/10.1016/j.matpr.2021.04.041.*
- *Lindenthal, T. (2014) "Valuable words: The price dynamics of internet domain names,"*
- *Journal of the Association for Information Science and Technology, 65(5), pp. 869–881. Available at: https://doi.org/10.1002/asi.23012.*
- *What is the K-nearest neighbors algorithm? (No date) IBM. IBM. Available at: https://www.ibm.com/uk-en/topics/knn#:~:text=The%20k%2Dnearest%20neighbors%20algorithm%2C%20also%20known%20as%20KNN%20or,of%20an%20individual%20data%20point. (Accessed: December 14, 2022).*
- Pedamkar, P. (2022) *Decision tree algorithm: Explanation and role of entropy in decision tree*, *EDUCBA*. Available at: https://www.educba.com/decision-tree-algorithm/ (Accessed: December 14, 2022).
- Richer, V. (2021) *Understanding decision trees (once and for all!)* 🤲, *Medium*. Towards Data Science. Available at: https://towardsdatascience.com/understanding-decision-trees-once-and-for-all-2d891b1be579 (Accessed: December 14, 2022).
- Breiman, L. (2001) *Machine Learning*, 45(1), pp. 5–32. Available at: https://doi.org/10.1023/a:1010933404324
- Scikit-learn: Machine Learning in Python, (2011) Pedregosa et al., JMLR 12, pp. 2825-2830.
- 1.6. nearest neighbors (no date) scikit. Available at: https://scikit-learn.org/stable/modules/neighbors.html (Accessed: December 15, 2022).
- Cutler, A. (2013) *Trees and random forests - Utah State University*. Available at: https://www.usu.edu/math/adele/RandomForests/UofU2013.pdf (Accessed: December 17, 2022).
- *1.10. decision trees* (no date) *scikit*. Available at: https://scikit-learn.org/stable/modules/tree.html (Accessed: December 15, 2022).
- Lyashenko, V. (2022) Cross-validation in Machine Learning: How To Do It Right, neptune.ai. Available at: https://neptune.ai/blog/cross-validation-in-machine-learning-how-to-do-it-right (Accessed: December 17, 2022).

Appendix 1:

Agreement of Participation – Group Assignment One CN7050

Please complete this agreement and keep a copy for each member of your group. The original of this agreement goes to your Tutor.

We agree to work as a group (group of 2-3) to complete the course work for CN7050 and understand that the grade awarded will be the grade allocated to us individually as a result of our group work.

Student No.	Name (block letters) and e-Mail Address	Signature
u2235955	Aekkaraj Kuplakatee Email: u2235955@uel.ac.uk	
u2228551	Adeyemi Farooq Opeyemi Email: u2228551@uel.ac.uk	
u2337009	Prabhakar Kenneth Vemagiri Email: u2337009@uel.ac.uk	

Note: Students should form their groups (group of 2-3) within the SAME Tutorial / Practical.

Tutorial / Practical Number: CN7050

Tutor's Name: Dr Mustansar Ghazanfar

Date of agreement: 12 Dec. 22

Appendix 2:

Install and Import libaraies

In [1]:
```
# Install libraries
!pip3 install python-whois
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting python-whois
  Downloading python-whois-0.8.0.tar.gz (109 kB)
     |████████████████████████████████| 109 kB 5.3 MB/s
Requirement already satisfied: future in /usr/local/lib/python3.8/dist-packages (from python-whois) (0.16.0)
Building wheels for collected packages: python-whois
  Building wheel for python-whois (setup.py) ... done
  Created wheel for python-whois: filename=python_whois-0.8.0-py3-none-any.whl size=103262 sha256=fa1a93a7a8ebb
d3cf84bf0092f05d84d4dc02991d3ce6888e8f50239c5d766c3
  Stored in directory: /root/.cache/pip/wheels/24/20/6b/5550a3d6bef09ddaed74eb84006fc3d53f94867f1916794df4
Successfully built python-whois
Installing collected packages: python-whois
Successfully installed python-whois-0.8.0
```

In [2]:
```
# Import libraries
import numpy as np
import pandas as pd
from google.colab import drive
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import GenericUnivariateSelect
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn import metrics
import matplotlib.pyplot as plt
import whois
import datetime
import seaborn as sns
import re
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

Load Data

In [3]:
```
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [4]:
```
# import the dataset
DF = pd.read_csv('/content/drive/MyDrive/CN7050/coursework_data.csv')

print(DF)
```

```
                    Domain  category
0        noriomatsumoto.com         1
1          haplessmales.com         1
2       shivamchatuvedi.com         1
3     dailyfreebieoffer.com         1
4             poniemall.com         1
...                     ...       ...
9995          citymark.com         5
9996        offroading.com         5
9997            cevon.com          5
9998          mozzilla.com         5
9999             stemz.com         5

[10000 rows x 2 columns]
```

Features Extraction

In [5]:
```
# Extrat lens of the domain
DF['Length'] = DF['Domain'].astype(str).str.len()

# Extract special characters from the domians
DF["Special_chars"] = DF.apply(lambda p: sum(not q.isalpha() and not q.isdigit() for q in p["Domain"]), axis=1)

# Extrct letters from the domains including Uppercase and Lowercase
DF["Letters"] = DF.apply(lambda p: sum(q.isalpha() for q in p["Domain"]), axis=1)
DF['Upper_case'] = DF.apply(lambda p: sum(q.isupper() for q in p['Domain']), axis=1)
DF['Lower_case'] = DF.apply(lambda p: sum(q.islower() for q in p['Domain']), axis=1)

# Separate the letters to vowels and consonants
DF['Vowels'] = DF.Domain.str.lower().str.count(r'[aeiou]')
DF['Consonants'] = DF.Domain.str.lower().str.count(r'[a-z]') - DF['Vowels']

# Extract numbers from the domains
DF["Numbers"] = DF.apply(lambda p: sum(q.isdigit() for q in p["Domain"]), axis=1)

DF.head(100)
```

Out[5]:

	Domain	category	Length	Special_chars	Letters	Upper_case	Lower_case	Vowels	Consonants	Numbers
0	noriomatsumoto.com	1	18	1	17	0	17	8	9	0
1	haplessmales.com	1	16	1	15	0	15	5	10	0
2	shivamchatuvedi.com	1	19	1	18	0	18	7	11	0
3	dailyfreebieoffer.com	1	21	1	20	0	20	9	11	0
4	poniemall.com	1	13	1	12	0	12	5	7	0
...	...	...	...	...	...	...	...	...	...	...
95	purewhitekidneybeans.com	1	24	1	23	0	23	9	14	0
96	yq83738128.com	1	14	1	5	0	5	1	4	8
97	cc549888.com	1	12	1	5	0	5	1	4	6
98	oatzone.com	1	11	1	10	0	10	5	5	0
99	bluelightblocka.com	1	19	1	18	0	18	6	12	0

100 rows × 10 columns

In [6]:
```
# Create the function to perform the extraction and use it with the fucntions down below.
def perform_whois(Domain):
    try:
        re = whois.whois(Domain)
        return re
    except Exception:
        return False
```

In [7]:
```
# Create a function to extract the register date from the domains.
def check_registered_date(re):
    if(re!=False):
        RD = re.creation_date
        if((RD is not None) and (type(RD)!=str)):
            if(type(RD)==list):
                RD=RD[0]
            TD = datetime.datetime.now()
            days = (TD-RD).days
            return days
        else:
            return 0
    else:
        return 0
```

In [8]:
```
# Create the fucntion to extract the expire date from the domains.
def check_expiration_date(re):
    if(re!=False):
        ED = re.expiration_date
        if((ED is not None) and (type(ED)!=str)):
            if(type(ED)==list):
                ED = ED[0]
            TD = datetime.datetime.now()
            days = (TD-ED).days
            return days
        else:
            return 0
    else:
        return 0
```

In [9]:
```
# Create the function to extrct the update date from the domains.
def check_updated_date(re):
    if(re!=False):
        UDD = re.updated_date
        if((UDD is not None) and (type(UDD)!=str)):
            if(type(UDD)==list):
                UDD = UDD[0]
            TD = datetime.datetime.now()
            days = (TD-UDD).days
            return days
        else:
            return 0
    else:
        return 0
```

In [10]:
```
# Crrate the function that can extract all thress fucntions above at the same time.
registered_date = []
expiration_date = []
updated_date = []

def extract_features():
    counter = 0
    for domain in DF['Domain']:
        counter = counter + 1
        whois_result = perform_whois(domain)
        registered_date.append(check_registered_date(whois_result))
        expiration_date.append(check_expiration_date(whois_result))
        updated_date.append(check_updated_date(whois_result))
    print(counter)
```

In [11]:
```
# extract features
extract_features()
```

```
Error trying to connect to socket: closing socket - [Errno -2] Name or service not known
10000
```

In [12]:
```
# Add the the feature that we extracted to dataset.
DF['reg_Date'] = registered_date
DF['exp_Date'] = expiration_date
DF['updated_Date'] = updated_date

DF.head()
```

Out[12]:

	Domain	category	Length	Special_chars	Letters	Upper_case	Lower_case	Vowels	Consonants	Numbers	reg_Date	exp_Date	updated_Date
0	noriomatsumoto.com	1	18	1	17	0	17	8	9	0	332	-33	332
1	haplessmales.com	1	16	1	15	0	15	5	10	0	0	0	0
2	shivamchatuvedi.com	1	19	1	18	0	18	7	11	0	0	0	0
3	dailyfreebieoffer.com	1	21	1	20	0	20	9	11	0	0	0	0
4	poniemall.com	1	13	1	12	0	12	5	7	0	0	0	0

Data Pre - Processing

In [13]:
```
# Use one-hot encoding to covert string column to numeric
domian_encoder = OneHotEncoder()
domian_encoder = domian_encoder.fit_transform(DF.Domain.values.reshape(-1,1)).toarray()
OneHot = pd.DataFrame(domian_encoder, columns = ["Domain_"+str(int(i)) for i in range(domian_encoder.shape[1])])

# Add the string tranformed to dataset
DF = pd.concat([DF, OneHot], axis=1)
DF = DF.drop('Domain', axis=1)
DF.head(100)
```

Out[13]:

	category	Length	Special_chars	Letters	Upper_case	Lower_case	Vowels	Consonants	Numbers	reg_Date	...	Domain_9990	Domain_9991	Domain_999
0	1	18	1	17	0	17	8	9	0	332	...	0.0	0.0	0.(
1	1	16	1	15	0	15	5	10	0	0	...	0.0	0.0	0.(
2	1	19	1	18	0	18	7	11	0	0	...	0.0	0.0	0.(
3	1	21	1	20	0	20	9	11	0	0	...	0.0	0.0	0.(
4	1	13	1	12	0	12	5	7	0	0	...	0.0	0.0	0.(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
95	1	24	1	23	0	23	9	14	0	0	...	0.0	0.0	0.(
96	1	14	1	5	0	5	1	4	8	0	...	0.0	0.0	0.(
97	1	12	1	5	0	5	1	4	6	0	...	0.0	0.0	0.(
98	1	11	1	10	0	10	5	5	0	0	...	0.0	0.0	0.(
99	1	19	1	18	0	18	6	12	0	0	...	0.0	0.0	0.(

100 rows × 10012 columns

In [14]:
```
# Spllt the data to X and Y
x = DF.drop('category', axis=1)
y = DF['category']

print("X = ","\n", x)
print('----------------------------------------------------------------------')
print("Y = ","\n", y)
```

```
X =
       Length  Special_chars  Letters  Upper_case  Lower_case  Vowels  \
0          18              1       17           0          17       8
1          16              1       15           0          15       5
2          19              1       18           0          18       7
3          21              1       20           0          20       9
4          13              1       12           0          12       5
...       ...            ...      ...         ...         ...     ...
9995       12              1       11           0          11       3
9996       14              1       13           0          13       5
9997        9              1        8           0           8       3
9998       12              1       11           0          11       4
9999        9              1        8           0           8       2

      Consonants  Numbers  reg_Date  exp_Date  ...  Domain_9990  Domain_9991  \
0              9        0       332       -33  ...          0.0          0.0
1             10        0         0         0  ...          0.0          0.0
2             11        0         0         0  ...          0.0          0.0
3             11        0         0         0  ...          0.0          0.0
4              7        0         0         0  ...          0.0          0.0
...          ...      ...       ...       ...  ...          ...          ...
9995           8        0      6142       -67  ...          0.0          0.0
9996           8        0      8740      -391  ...          0.0          0.0
9997           5        0      6179     -1856  ...          0.0          0.0
9998           7        0      7326      -344  ...          0.0          0.0
9999           6        0      6611      -328  ...          0.0          0.0

      Domain_9992  Domain_9993  Domain_9994  Domain_9995  Domain_9996  \
0             0.0          0.0          0.0          0.0          0.0
1             0.0          0.0          0.0          0.0          0.0
2             0.0          0.0          0.0          0.0          0.0
3             0.0          0.0          0.0          0.0          0.0
4             0.0          0.0          0.0          0.0          0.0
...           ...          ...          ...          ...          ...
9995          0.0          0.0          0.0          0.0          0.0
9996          0.0          0.0          0.0          0.0          0.0
9997          0.0          0.0          0.0          0.0          0.0
9998          0.0          0.0          0.0          0.0          0.0
9999          0.0          0.0          0.0          0.0          0.0

      Domain_9997  Domain_9998  Domain_9999
0             0.0          0.0          0.0
1             0.0          0.0          0.0
2             0.0          0.0          0.0
3             0.0          0.0          0.0
4             0.0          0.0          0.0
...           ...          ...          ...
9995          0.0          0.0          0.0
9996          0.0          0.0          0.0
9997          0.0          0.0          0.0
9998          0.0          0.0          0.0
9999          0.0          0.0          0.0

[10000 rows x 10011 columns]
----------------------------------------------------------------------
Y =
 0       1
1       1
2       1
3       1
4       1
       ..
9995    5
9996    5
9997    5
9998    5
9999    5
Name: category, Length: 10000, dtype: int64
```

In [15]:
```
# Perform feature selection.
transform = GenericUnivariateSelect(mode="fpr")
gus = transform.fit_transform(x,y)

print(gus)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/feature_selection/_univariate_selection.py:112: UserWarning: Fea
tures [3] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx, UserWarning)
/usr/local/lib/python3.8/dist-packages/sklearn/feature_selection/_univariate_selection.py:113: RuntimeWarning:
invalid value encountered in true_divide
  f = msb / msw
```

```
[[ 1.800e+01  1.000e+00  1.700e+01 ...  3.320e+02 -3.300e+01  3.320e+02]
 [ 1.600e+01  1.000e+00  1.500e+01 ...  0.000e+00  0.000e+00  0.000e+00]
 [ 1.900e+01  1.000e+00  1.800e+01 ...  0.000e+00  0.000e+00  0.000e+00]
 ...
 [ 9.000e+00  1.000e+00  8.000e+00 ...  6.179e+03 -1.856e+03  1.526e+03]
 [ 1.200e+01  1.000e+00  1.100e+01 ...  7.326e+03 -3.440e+02  4.190e+02]
 [ 9.000e+00  1.000e+00  8.000e+00 ...  6.611e+03 -3.280e+02  3.700e+01]]
```

In [16]:
```
# Spilt X and Y to train and test the dataset.
x_train, x_test, y_train, y_test = train_test_split(gus, y, test_size = 0.3, stratify=y, random_state=0)

print('xTrain is\n', x_train, '\n', 'yTrain is\n', y_train)
print('-----------------------------------------------------------------')
print('xTest is\n', x_test,'\n', 'yTest is\n', y_test)
```

```
xTrain is
 [[ 2.200e+01  1.000e+00  2.100e+01 ...  4.750e+02 -2.550e+02  1.090e+02]
 [ 1.200e+01  1.000e+00  1.100e+01 ...  5.160e+02 -2.140e+02  1.880e+02]
 [ 1.700e+01  1.000e+00  1.600e+01 ...  6.251e+03 -3.230e+02  5.600e+01]
 ...
 [ 1.000e+01  1.000e+00  3.000e+00 ...  5.470e+02 -1.830e+02  1.650e+02]
 [ 1.600e+01  1.000e+00  1.500e+01 ...  0.000e+00  0.000e+00  0.000e+00]
 [ 1.200e+01  1.000e+00  1.100e+01 ...  3.766e+03 -2.510e+02  1.450e+02]]
 yTrain is
 1854    1
465     1
8707    3
853     1
4906    1
       ..
5793    1
4815    1
2498    1
4743    1
6757    2
Name: category, Length: 7000, dtype: int64
-----------------------------------------------------------------
xTest is
 [[ 1.200e+01  1.000e+00  1.100e+01 ...  4.260e+02 -3.040e+02  6.500e+01]
 [ 1.400e+01  1.000e+00  1.300e+01 ...  2.603e+03 -3.190e+02  2.400e+01]
 [ 1.500e+01  1.000e+00  1.400e+01 ...  5.540e+02 -1.760e+02  7.400e+01]
 ...
 [ 1.800e+01  1.000e+00  1.700e+01 ...  3.949e+03 -8.000e+02  8.400e+01]
 [ 1.100e+01  1.000e+00  1.000e+01 ...  2.030e+02 -1.620e+02  8.500e+01]
 [ 1.200e+01  1.000e+00  1.100e+01 ...  0.000e+00  0.000e+00  0.000e+00]]
 yTest is
 94      1
6879    2
2499    1
5485    1
4786    1
       ..
6333    2
4103    1
6377    2
4069    1
4272    1
Name: category, Length: 3000, dtype: int64
```

In [17]:
```
# Tranform the dataset to a common standad scale by using the Standard Scaler
standardscaler = StandardScaler()
x_train = standardscaler.fit_transform(x_train)
x_test = standardscaler.transform(x_test)

print('xTrain after scalar is\n', x_train)
print('----------------------------------------------------------------')
print('xTest after scalar is\n', x_test)
```

```
xTrain after scalar is
 [[ 1.22916764 -0.20750887  1.23550236 ... -0.58865731 -0.10988759
   0.09948588]
 [-0.77707516 -0.20750887 -0.64436473 ... -0.57562029 -0.01211556
   0.47275705]
 [ 0.22604624 -0.20750887  0.29556882 ...  1.24797253 -0.27204608
  -0.15093655]
 ...
 [-1.17832372 -0.20750887 -2.1482584  ... -0.56576303  0.06180964
   0.36408316]
 [ 0.02542196 -0.20750887  0.10758211 ... -0.73969594  0.49820676
  -0.41553384]
 [-0.77707516 -0.20750887 -0.64436473 ...  0.45780197 -0.10034885
   0.26958413]]
----------------------------------------------------------------
xTest after scalar is
 [[-0.77707516 -0.20750887 -0.64436473 ... -0.60423813 -0.22673709
  -0.10841199]
 [-0.3758266  -0.20750887 -0.26839131 ...  0.08799579 -0.26250735
  -0.302135  ]
 [-0.17520232 -0.20750887 -0.0804046  ... -0.5635372   0.07850243
  -0.06588742]
 ...
 [ 0.42667052 -0.20750887  0.48355553 ...  0.5159916  -1.40954022
  -0.01863791]
 [-0.97769944 -0.20750887 -0.83235144 ... -0.6751468   0.111888
  -0.01391296]
 [-0.77707516 -0.20750887 -0.64436473 ... -0.73969594  0.49820676
  -0.41553384]]
```

Make Prediction

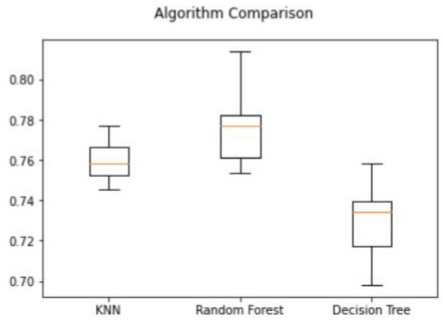
In [18]:
```
# Make prediction without parameters using K-Fold and Cross-Validation.
mlModels = []
mlModels.append(('KNN', KNeighborsClassifier()))
mlModels.append(('Random Forest', RandomForestClassifier()))
mlModels.append(('Decision Tree', DecisionTreeClassifier()))

models = []
results = []

for name, model in mlModels:
  kFold = KFold(n_splits=15, shuffle = True, random_state=123)
  cvsResults = cross_val_score(model, x_train, y_train, cv=kFold, scoring='accuracy')
  results.append(cvsResults)
  models.append(name)
  print("%s: %f (%f)" % (name, cvsResults.mean(), cvsResults.std()))
```

```
KNN: 0.759004 (0.008914)
Random Forest: 0.775856 (0.018080)
Decision Tree: 0.729006 (0.016139)
```

In [19]:
```
# Visualisation for the K-Fold and Cross-Validation
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(models)
plt.show()
```



In [20]:
```
# Check the k-values that have the lowest error
def k_value_error():
  error_rate=[]
  for i in range (1,31):
    clf=KNeighborsClassifier(n_neighbors=i)
    clf.fit(x_train,y_train)
    predict_i=clf.predict(x_test)
    error_rate.append(np.mean(predict_i!=y_test))
    error_rate
    k_value = error_rate.index(min(error_rate)) + 1

# Plotting the error rate vs k range
  plt.figure(figsize=(12,6))
  plt.plot(range(1,31),error_rate,marker="o",markerfacecolor="red",
           linestyle="solid",color="green",markersize=15)
  plt.title("Error rate vs k value",fontsize=20)
  plt.xlabel("k-values",fontsize=20)
  plt.ylabel("error rate",fontsize=20)
  plt.xticks(range(1,31))
  plt.show()
  return k_value
```

In [21]:
```
# Check the max_depth that have the lowest error for Random Forest
def m_max_depth_for_RF():
  error_rate=[]
  for m in range (1,31):
    rfm=RandomForestClassifier(max_depth=m)
    rfm.fit(x_train,y_train)
    predict_i=rfm.predict(x_test)
    error_rate.append(np.mean(predict_i!=y_test))
    error_rate
    depth = error_rate.index(min(error_rate)) + 1

# Plotting the error rate vs max depth
  plt.figure(figsize=(12,6))
  plt.plot(range(1,31),error_rate,marker="o",markerfacecolor="yellow",
          linestyle="solid",color="darkblue",markersize=15)
  plt.title("Error rate vs max depth for random forest",fontsize=20)
  plt.xlabel("max_depth",fontsize=20)
  plt.ylabel("error rate",fontsize=20)
  plt.xticks(range(1,31))
  plt.show()
  return depth
```

In [22]:
```
# Check the max_depth that have the lowest error for Decision Tree
def m_max_depth_for_DT():
  error_rate=[]
  for m in range (1,31):
    dtm=DecisionTreeClassifier(max_depth=m)
    dtm.fit(x_train,y_train)
    predict_i=dtm.predict(x_test)
    error_rate.append(np.mean(predict_i!=y_test))
    error_rate
    depth = error_rate.index(min(error_rate)) + 1

# Plotting the error rate vs max depth
  plt.figure(figsize=(12,6))
  plt.plot(range(1,31),error_rate,marker="o",markerfacecolor="green",
          linestyle="solid",color="brown",markersize=15)
  plt.title("Error rate vs max depth for decision tree",fontsize=20)
  plt.xlabel("max_depth",fontsize=20)
  plt.ylabel("error rate",fontsize=20)
  plt.xticks(range(1,31))
  plt.show()
  return depth
```

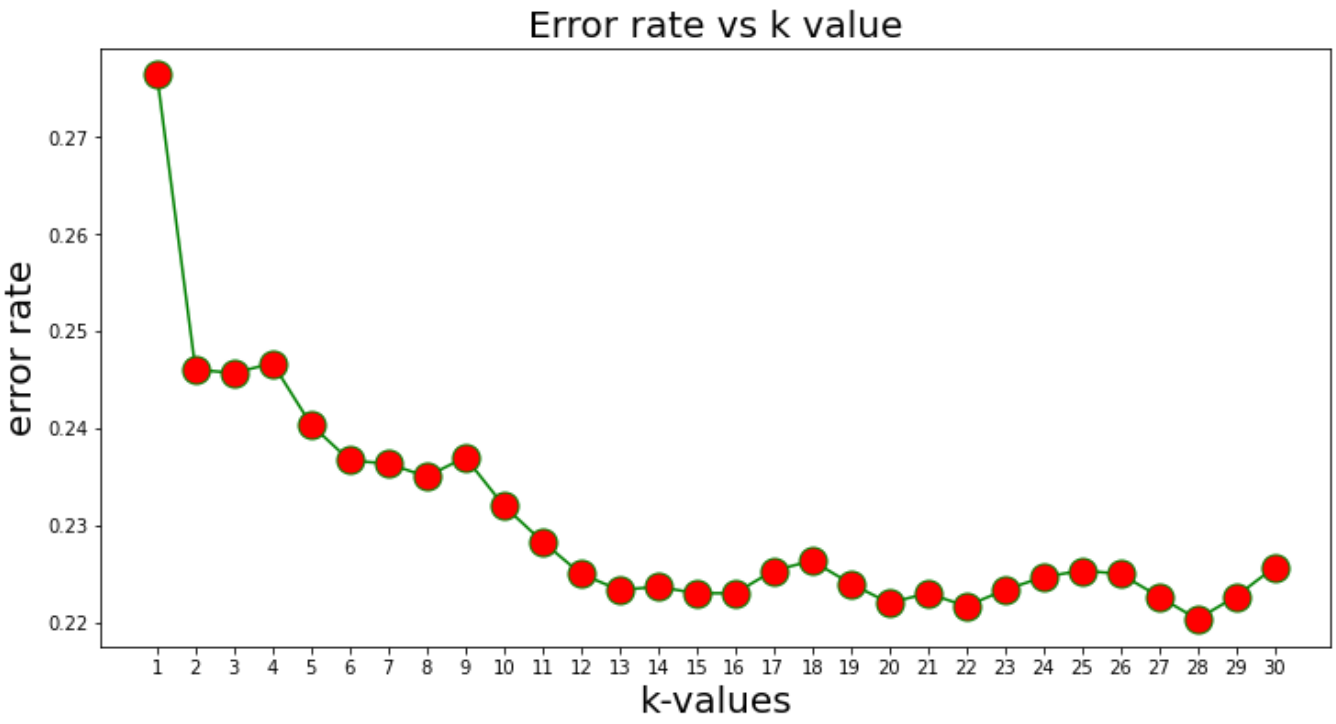
In [23]:
```
# Defind the K-Nearest Neighbours with parameters as a function to make a prediction. Also, make it easy to comp
are with other clssifications.
def KNN_result(k_value_error):
  knn = KNeighborsClassifier(n_neighbors=k_value_error)
  knn_model = knn.fit(x_train,y_train)
  y_pred = knn_model.predict(x_test)
  ac = accuracy_score(y_test, y_pred)
  conf = confusion_matrix(y_test, y_pred)
  accuracy = classification_report(y_test, y_pred)
  result = {"accuracy":accuracy, "ac":ac, "confusion matrix":conf}
  return result
```

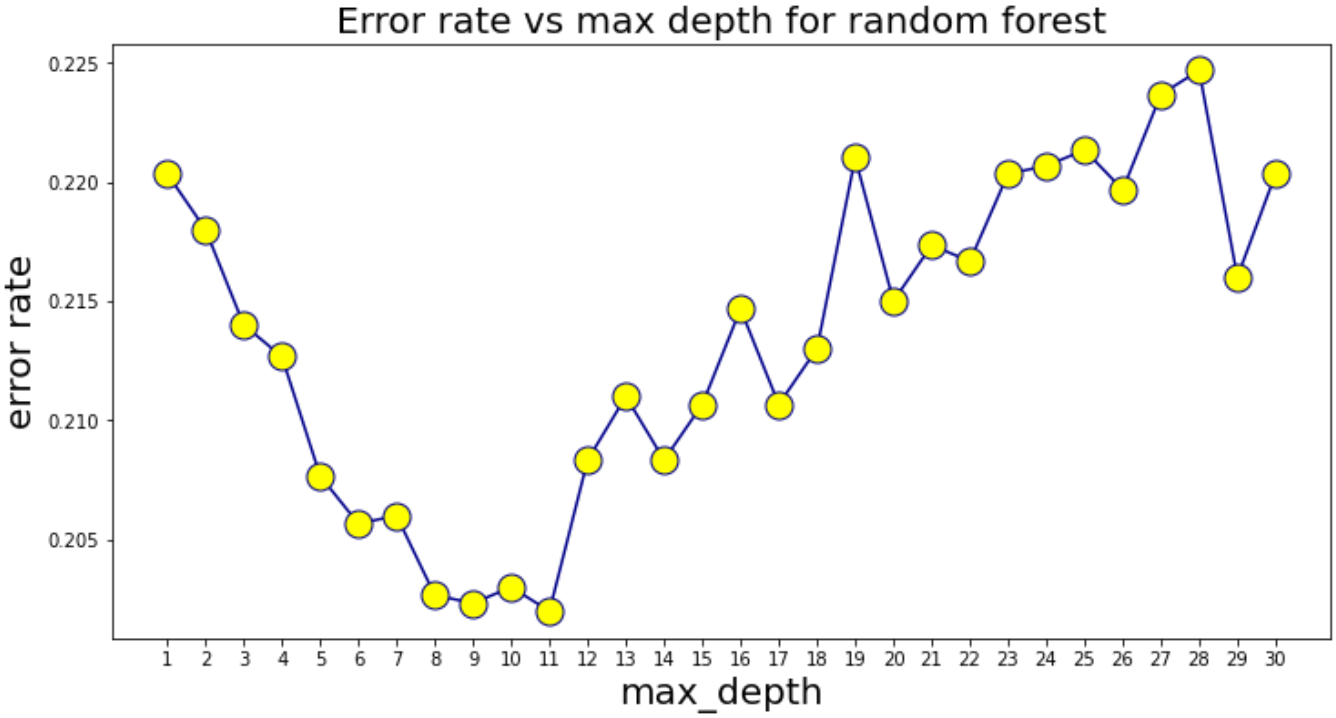
In [24]:
```
# Defind the Random Forest with parameters as a function to make a prediction. Also, make it easy to compare wit
h other clssifications.
def RF_result(m_max_depth_for_RF):
  rf = RandomForestClassifier(max_depth=m_max_depth_for_RF,random_state=0)
  rf_model = rf.fit(x_train, y_train)
  y_pred = rf_model.predict(x_test)
  ac = accuracy_score(y_test, y_pred)
  conf = confusion_matrix(y_test, y_pred)
  accuracy = classification_report(y_test, y_pred, zero_division=1)
  result = {"accuracy":accuracy, "ac":ac, "confusion matrix":conf}
  return result
```

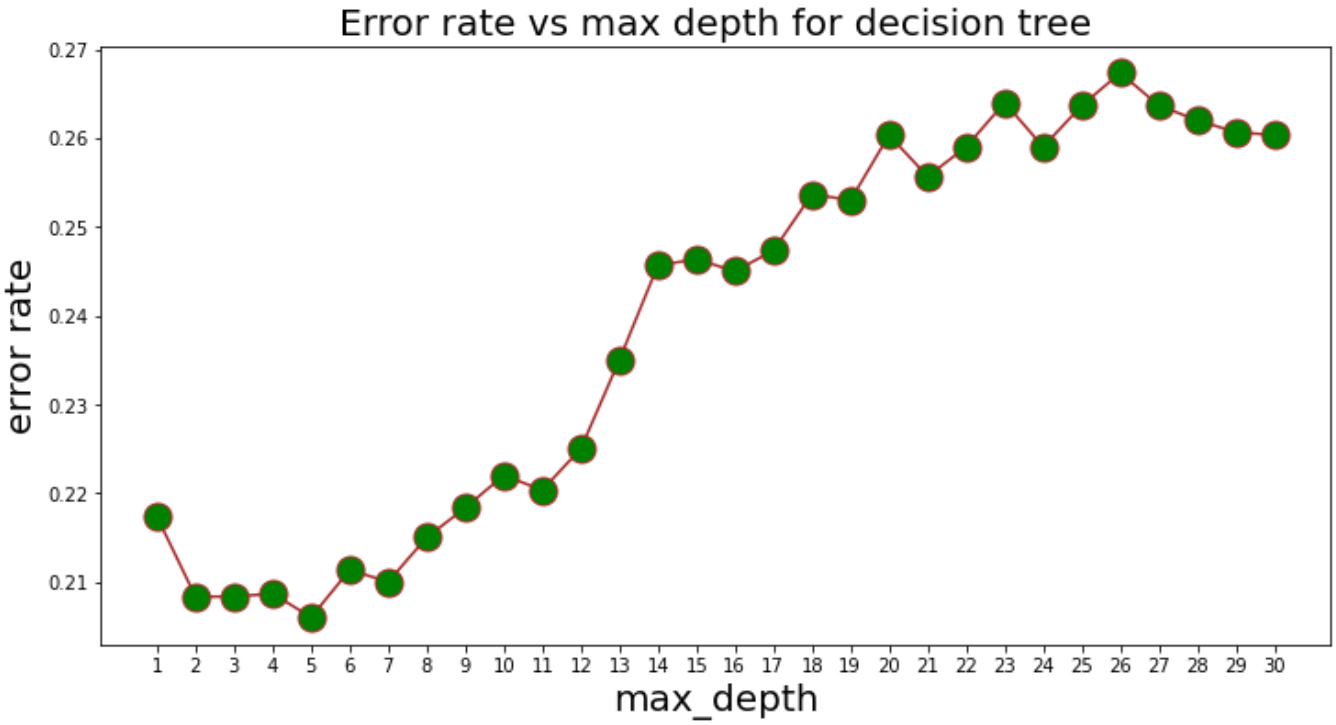
In [25]:
```
# Defind the Decision Tree with parameters as a function to make a prediction. Also, make it easy to compare wit
h other clssifications.
def DT_result(m_max_depth_for_DT):
  dt = DecisionTreeClassifier(max_depth=m_max_depth_for_DT)
  dt_model = dt.fit(x_train,y_train)
  y_pred = dt_model.predict(x_test)
  ac = accuracy_score(y_test, y_pred)
  conf = confusion_matrix(y_test, y_pred)
  accuracy = classification_report(y_test, y_pred, zero_division=1)
  result = {"accuracy":accuracy, "ac":ac, "confusion matrix":conf}
  return result
```

In [26]:
```
# Defind function to extract all the result from the classification functions above.
def classification_results():
    results = {}
    knn = KNN_result(k_value_error())
    rf = RF_result(m_max_depth_for_RF())
    dt = DT_result(m_max_depth_for_DT())
    results = {"K-Nearset Neighbors":knn,
               "Random Forest":rf,
               "Decision Tree":dt
               }
    return results
```

In [27]:
```
# Show the result of classification with parameters
classification_results = classification_results()
```




Error rate vs max depth for random forest
error rate
max_depth
0.225
0.220
0.215
0.210
0.205
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30


Error rate vs max depth for decision tree
error rate
max_depth
0.27
0.26
0.25
0.24
0.23
0.22
0.21
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

In [28]:
```
# Print out the classification results
print("----------Accuracy----------")
for k,r in classification_results.items():
    print(f"{k}: {r['ac']*100}","%")
print("\n----------Confusion matrix----------")
for k,r in classification_results.items():
     print(f"{k}:\n {r['confusion matrix']}")
print("\n----------Classification Report----------")
for k,r in classification_results.items():
     print(f"{k}:\n {r['accuracy']}",'%')
```

```
----------Accuracy----------
K-Nearset Neighbors: 77.96666666666667 %
Random Forest: 79.53333333333333 %
Decision Tree: 79.4 %

----------Confusion matrix----------
K-Nearset Neighbors:
 [[1784   16    0    0    0]
 [  75  487   22    4   12]
 [  37  223   22    7   11]
 [   7  114   13    2   14]
 [  10   74   12   10   44]]
Random Forest:
 [[1782   16    2    0    0]
 [  34  538   11    3   14]
 [  12  255   11    7   15]
 [   3  111    9    7   20]
 [   4   77   13    8   48]]
Decision Tree:
 [[1782   18    0    0    0]
 [  32  531   11    8   18]
 [   9  261    6    7   17]
 [   3  113    2    8   24]
 [   4   79    0   12   55]]


----------Classification Report----------
K-Nearset Neighbors:
              precision    recall  f1-score   support

           1       0.93      0.99      0.96      1800
           2       0.53      0.81      0.64       600
           3       0.32      0.07      0.12       300
           4       0.09      0.01      0.02       150
           5       0.54      0.29      0.38       150

    accuracy                           0.78      3000
   macro avg       0.48      0.44      0.43      3000
weighted avg       0.73      0.78      0.74      3000
 %
Random Forest:
              precision    recall  f1-score   support

           1       0.97      0.99      0.98      1800
           2       0.54      0.90      0.67       600
           3       0.24      0.04      0.06       300
           4       0.28      0.05      0.08       150
           5       0.49      0.32      0.39       150

    accuracy                           0.80      3000
   macro avg       0.50      0.46      0.44      3000
weighted avg       0.75      0.80      0.75      3000
 %
Decision Tree:
              precision    recall  f1-score   support

           1       0.97      0.99      0.98      1800
           2       0.53      0.89      0.66       600
           3       0.32      0.02      0.04       300
           4       0.23      0.05      0.09       150
           5       0.48      0.37      0.42       150

    accuracy                           0.79      3000
   macro avg       0.51      0.46      0.44      3000
weighted avg       0.76      0.79      0.75      3000
 %
```

In [29]:
```
# Visualisation for prediction with parameters on accuracy
accuracy = []
name = []
for k,r in classification_results.items():
    name.append(k)
    accuracy.append(r['ac'])
print(accuracy)
print(name)

fig, ax = plt.subplots(facecolor='w')
ax.bar(name, accuracy,
       color = 'lightblue',
       edgecolor = 'darkblue',
       width =0.6)
ax.autoscale(enable=True)
ax.set(title = "Compare Models Graphically",
       xlabel = "Models",
       ylabel = "Accuracy")
plt.show()
```

```
[0.7796666666666666, 0.7953333333333333, 0.794]
['K-Nearset Neighbors', 'Random Forest', 'Decision Tree']
```