Shri Vile Parle Kelavani Mandal
DWARKADAS J. SANGHVI
(Autonomous College Affiliated
NAAC Accredited with "A" Grade (
Department of Computer

ACL

Experiment 1

NAME: Divyesh Khunt **Sapid:60009210116** **Batch:D12**

Aim: - Implement a Spam classifier using Naïve Bayes classifier

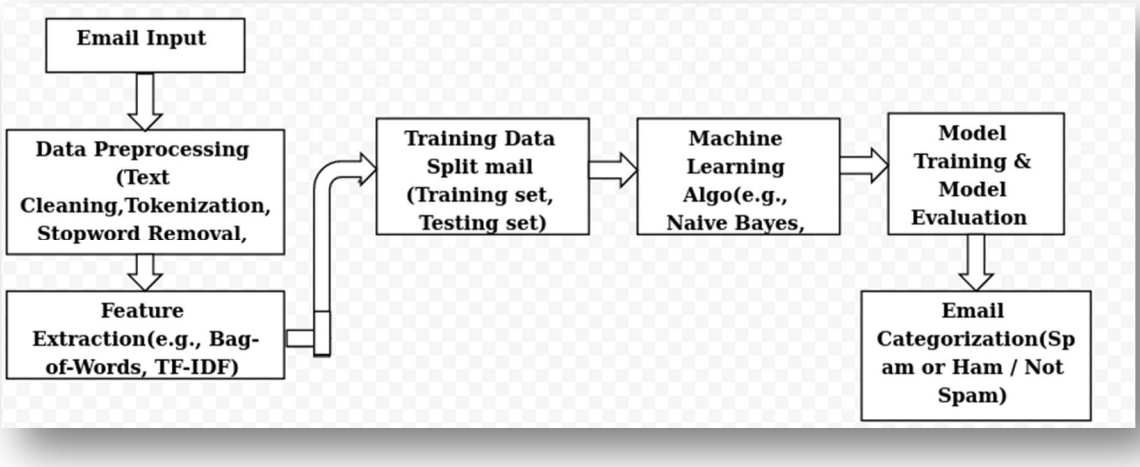
Theory:

Spam Classification

Spam classification, also known as email spam filtering or spam detection, is a classic problem in natural language processing and machine learning. It involves automatically identifying and categorizing incoming emails or messages as either "spam" (unsolicited and often irrelevant or inappropriate) or "ham" (legitimate and desired).

The primary goal of spam classification is to build a model that can accurately distinguish between spam and non-spam messages, thereby reducing the amount of unwanted and potentially harmful content that reaches users' inboxes. This is especially important considering the sheer volume of spam emails generated daily, as well as the potential security risks associated with phishing attempts and malicious content that may be contained in spam messages.

To solve the spam classification problem, machine learning algorithms are employed. Common approaches include the use of traditional algorithms like Naive Bayes, support vector machines (SVMs), decision trees, and more modern techniques like deep learning using neural networks.



Naïve Bayes classifier

The Naive Bayes classifier is a simple, yet effective probabilistic machine learning algorithm commonly used for classification tasks, particularly in natural language processing, spam filtering, sentiment analysis, and more. Despite its simplicity, Naive Bayes often performs surprisingly well and can be computationally efficient, making it a popular choice for certain applications.

The "Naive" in Naive Bayes comes from the assumption that the features (or attributes) used for classification are conditionally independent, given the class label. In other words, the model assumes that each feature contributes independently to the probability of a specific class. This is a simplification, and in reality, features might be correlated, but the assumption makes the algorithm computationally tractable.

How Naive Bayes works:

1. Data and Labels: The Naive Bayes classifier is trained on a labeled dataset containing features and their corresponding class labels. For example, in email spam classification, the features could be the words in the email, and the labels would be "spam" or "ham" (not spam).
2. Prior Probability: The first step in Naive Bayes is to calculate the prior probabilities of each class. The prior probability of a class is the probability of that class occurring without considering any features. For example, if you have 70% spam emails and 30% non-spam emails in your dataset, the prior probabilities would be P(spam) = 0.7 and P(ham) = 0.3.
3. Likelihood Estimation: Next, Naive Bayes calculates the likelihood probabilities for each feature given the class labels. The likelihood probability is the probability of observing a particular feature given the class. For example, P(word='free' | spam) would represent the probability of the word "free" appearing in spam emails.
4. Posterior Probability: Using Bayes' theorem, the Naive Bayes classifier calculates the posterior probability of each class given the observed features. The posterior probability is the probability of a class given the evidence (features). The class with the highest posterior probability is the predicted class for the input.
5. Prediction: The Naive Bayes classifier then makes predictions based on the highest posterior probability. The class with the highest posterior probability for a given set of features is assigned as the predicted class.

Lab Experiments to be Performed in This Session: -

Data Set: - SMS Spam Collection

This dataset contains SMS messages labeled as spam or ham. It is commonly used for spam detection in text messages.

Exercise 1: - Perform Spam Classification With text preprocessing steps.

Step 1: Import Labeled with labels as "spam" or "not spam" (ham).

Step 2: Perform Data Preprocessing to convert it into a suitable format for training the Naive Bayes classifier. Common steps include (Removing punctuation and special characters, Tokenization, stop word Removal, converting to lowercase, Stemming or Lemmatization)

Step 3: Feature Extraction (bag-of-words model or term frequency-inverse document frequency (TF-IDF) representation.

Step 4: Training the Naive Bayes Classifier

Step 5: Evaluate the performance of the trained Naive Bayes classifier on the test dataset.

Step 6: Hyperparameter Tuning (Optional)

Step 7: Deployment Once you are satisfied with the model's performance, you can deploy it to classify new, unseen emails or messages as spam or ham.

Exercise 2: - Perform Spam Classification Without Using Any text preprocessing steps.

Step 1: - Import Labeled with labels as "spam" or "not spam" (ham).

Step 2: Feature Extraction (bag-of-words model or term frequency-inverse document frequency (TF-IDF) representation.

Step 3: Training the Naive Bayes Classifier

Step 4: Evaluate the performance of the trained Naive Bayes classifier on the test dataset.

Step 5: Hyperparameter Tuning (Optional)

Step 6: Deployment Once you are satisfied with the model's performance, you can deploy it to classify new, unseen emails or messages as spam or ham.

Exercise 3: -

a. Compare The results of Exercise 1 and Exercise 2
b. Show the Comparison using different visualization techniques.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
[ ] data=pd.read_csv("/content/spam.csv",encoding='latin-1')
    data.head()
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

```
[ ] data = data.drop(['Unnamed: 2','Unnamed: 3','Unnamed: 4'],axis=1)
    data
```

	v1	v2
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
...	...	...

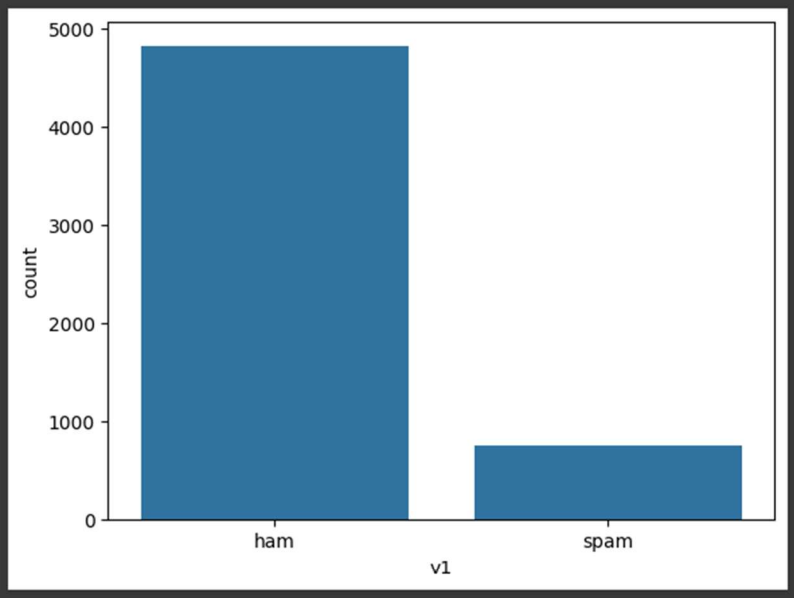
```
[ ] data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   v1      5572 non-null   object
 1   v2      5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

```
[ ] data['label_enc'] = data['v1'].map({'ham':0,'spam':1})
    data.head()
```

	v1	v2	label_enc
0	ham	Go until jurong point, crazy.. Available only ...	0
1	ham	Ok lar... Joking wif u oni...	0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	1
3	ham	U dun say so early hor... U c already then say...	0
4	ham	Nah I don't think he goes to usf, he lives aro...	0

```
sns.countplot(x=data['v1'])
plt.show()
```




```
[ ] #!pip install num2words
    import nltk
    from nltk.tokenize import word_tokenize
    from nltk.corpus import stopwords
    from nltk.tokenize import word_tokenize
    from nltk.stem import PorterStemmer, LancasterStemmer
    from nltk.stem import WordNetLemmatizer
```

```
Requirement already satisfied: num2words in /usr/local/lib/python3.10/dist-packages (0.5.13)
Requirement already satisfied: docopt>=0.6.2 in /usr/local/lib/python3.10/dist-packages (from num2words) (0.6.2)
```

```
[ ] nltk.download('punkt')
    nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
import string
def preprocess_text(text):

    text = text.translate(str.maketrans('', '', string.punctuation))

    tokens = word_tokenize(text)

    tokens = [token.lower() for token in tokens]

    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stop_words]

    stemmer = PorterStemmer()
    tokens = [stemmer.stem(token) for token in tokens]

    return ' '.join(tokens)
```

```
[ ] messages = []
    for i in range(0, len(data)):
        message = preprocess_text(data.v2[i])
        messages.append(message)
```

```
message=data.v2[0]
message
```

```
'Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...'
```

```
messages[0]
```

```
'go jurong point crazi avail bugi n great world la e buffet cine got amor wat'
```

```
clean_data = data.drop(["v2"],axis=1)
clean_data['messages'] = messages
clean_data.head()
```

	v1	label_enc	messages
0	ham	0	go jurong point crazi avail bugi n great world...
1	ham	0	ok lar joke wif u oni
2	spam	1	free entri 2 wkli comp win fa cup final tkt 21...
3	ham	0	u dun say earli hor u c alreadi say
4	ham	0	nah dont think goe usf live around though

NAVIE BAYES

```
y = data["v1"].values
x = data["v2"].values
```

```
from sklearn.model_selection import train_test_split
xtrain , xtest , ytrain , ytest = train_test_split(x,y, test_size = 0.2, random_state = 1)

print(xtrain.shape, ytrain.shape, xtest.shape, ytest.shape)
```

```
(4457,) (4457,) (1115,) (1115,)
```

```
xtrain
```

```
array(['Sleeping nt feeling well',
       'Come aftr  &lt;DECIMAL&gt; ..now i m cleaning the house',
       'Almost there, see u in a sec', ...,
       'Huh i cant thk of more oredi how many pages do we have?',
       'I have printed it oh. So  &lt;#&gt;  come upstairs',
       'K k:) sms chat with me.'], dtype=object)
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
vect = TfidfVectorizer(stop_words='english',max_df=0.5)

x_train = vect.fit_transform(xtrain)

x_test = vect.transform(xtest)

from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB()

nb.fit(x_train,ytrain)

y_pred_test = nb.predict(x_test)
y_pred_train = nb.predict(x_train)

from sklearn.metrics import accuracy_score
print(accuracy_score(ytest,y_pred_test)*100)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(ytest,y_pred_test)
print(cm)
```

```
98.02690582959642
[[976   0]
 [ 22 117]]
```

```
y = clean_data["v1"].values
x = clean_data["messages"].values
```

```
[ ] from sklearn.model_selection import train_test_split
    xtrain , xtest , ytrain , ytest = train_test_split(x,y, test_size = 0.2, random_state = 1)

    print(xtrain.shape, ytrain.shape, xtest.shape, ytest.shape)
```

```
(4457,) (4457,) (1115,) (1115,)
```

```
[ ] xtrain
```

```
array(['sleep nt feel well', 'come aftr ltdecimalgt clean hous',
       'almost see u sec', ..., 'huh cant thk oredi mani page',
       'print oh ltgt come upstair', 'k k sm chat'], dtype=object)
```

```
[ ] from sklearn.feature_extraction.text import TfidfVectorizer
    vect = TfidfVectorizer(stop_words='english',max_df=0.5)

    x_train = vect.fit_transform(xtrain)

    x_test = vect.transform(xtest)

    from sklearn.naive_bayes import MultinomialNB
    nb = MultinomialNB()

    nb.fit(x_train,ytrain)

    y_pred_test = nb.predict(x_test)
    y_pred_train = nb.predict(x_train)

    from sklearn.metrics import accuracy_score
    print(accuracy_score(ytest,y_pred_test)*100)

    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(ytest,y_pred_test)
    print(cm)
```

```
97.57847533632287
[[976   0]
 [ 27 112]]
```

logistic Regression

```
[ ] from sklearn.feature_extraction.text import TfidfVectorizer
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

    X = data['v2']
    y = data['label_enc']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    tfidf = TfidfVectorizer(stop_words='english')
    X_train_tfidf = tfidf.fit_transform(X_train)
    X_test_tfidf = tfidf.transform(X_test)

    model = LogisticRegression()
    model.fit(X_train_tfidf, y_train)

    y_pred = model.predict(X_test_tfidf)

    accuracy = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)

    print(f'Accuracy: {accuracy:.4f}')
    print('Confusion Matrix:')
    print(conf_matrix)
```

```
Accuracy: 0.9525
Confusion Matrix:
[[962   3]
 [ 50 100]]
```

```
    X = clean_data['messages']
    y = data['label_enc']


    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42

    tfidf = TfidfVectorizer(stop_words='english')
    X_train_tfidf = tfidf.fit_transform(X_train)
    X_test_tfidf = tfidf.transform(X_test)

    model = LogisticRegression()
    model.fit(X_train_tfidf, y_train)

    y_pred = model.predict(X_test_tfidf)

    accuracy = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)

    print(f'Accuracy: {accuracy:.4f}')
    print('Confusion Matrix:')
    print(conf_matrix)
```

```
Accuracy: 0.9507
Confusion Matrix:
[[963   2]
 [ 53  97]]
```

Random Forest

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier

X = data['v2']
y = data['v1']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

tfidf = TfidfVectorizer(stop_words='english')
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train_tfidf, y_train)

y_pred = model.predict(X_test_tfidf)

accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f'Accuracy: {accuracy:.4f}')
print('Confusion Matrix:')
print(conf_matrix)
```

```
Accuracy: 0.9767
Confusion Matrix:
[[964   1]
 [ 25 125]]
```

```
X = clean_data['messages']
y = clean_data['v1']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

tfidf = TfidfVectorizer(stop_words='english')
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train_tfidf, y_train)

y_pred = model.predict(X_test_tfidf)

accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f'Accuracy: {accuracy:.4f}')
print('Confusion Matrix:')
print(conf_matrix)
```

```
Accuracy: 0.9722
Confusion Matrix:
[[964   1]
 [ 30 120]]
```

```
[ ]  from sklearn.metrics import accuracy_score,confusion_matrix,precision_score
     from sklearn.linear_model import LogisticRegression
     from sklearn.svm import SVC
     from sklearn.naive_bayes import MultinomialNB
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.ensemble import AdaBoostClassifier
     from sklearn.ensemble import BaggingClassifier
     from sklearn.ensemble import ExtraTreesClassifier
     from sklearn.ensemble import GradientBoostingClassifier
```

```
[ ]  svc = SVC(kernel='sigmoid', gamma=1.0)
     knc = KNeighborsClassifier()
     mnb = MultinomialNB()
     dtc = DecisionTreeClassifier(max_depth=5)
     lrc = LogisticRegression(solver='liblinear', penalty='l1')
     rfc = RandomForestClassifier(n_estimators=50, random_state=2)
     abc = AdaBoostClassifier(n_estimators=50, random_state=2)
     bc = BaggingClassifier(n_estimators=50, random_state=2)
     gbdt = GradientBoostingClassifier(n_estimators=50,random_state=2)
     xgb = XGBClassifier(n_estimators=50,random_state=2)
```

```
[ ]  clfs = {
         'SVC' : svc,
         'KN' : knc,
         'MNB': mnb,
         'DT': dtc,
         'LR': lrc,
         'RF': rfc,
         'AdaBoost': abc,
         'BgC': bc,
         'GBDT':gbdt,
         'xgb':xgb
     }
```

```
[ ] def train_classifier(clf,X_train,y_train,X_test,y_test):
        clf.fit(X_train,y_train)
        y_pred = clf.predict(X_test)
        accuracy = accuracy_score(y_test,y_pred)
        precision = precision_score(y_test,y_pred)

        return accuracy,precision
```

```
[ ] y = data["v1"].values
    x = data["v2"].values
```

```
[ ] from sklearn.model_selection import train_test_split
    xtrain , xtest , ytrain , ytest = train_test_split(x,y, test_size = 0.2, random_state = 1)

    print(xtrain.shape, ytrain.shape, xtest.shape, ytest.shape)
```

```
(4457,) (4457,) (1115,) (1115,)
```

```
[ ] from sklearn.feature_extraction.text import TfidfVectorizer
    from sklearn.preprocessing import LabelEncoder

    vectorizer = TfidfVectorizer()
    X_train_vectorized = vectorizer.fit_transform(xtrain)
    X_test_vectorized = vectorizer.transform(xtest)

    label_encoder = LabelEncoder()
    y_train_encoded = label_encoder.fit_transform(ytrain)
    y_test_encoded = label_encoder.transform(ytest)

    accuracy_scores1 = []
    precision_scores1 = []
```

```python
for name, clf in clfs.items():
    current_accuracy,current_precision  = train_classifier(clf, X_train_vectorized, y_train_encoded, X_test_vectorized, y_test_encoded)

    print(name)
    print("Accuracy - ", current_accuracy)
    print("Precision - ",current_precision)

    accuracy_scores.append(accuracy_scores1)
    accuracy_scores.append(precision_scores1)
```

```
SVC
Accuracy -  0.9910313901345291
Precision -  0.9849624060150376
KN
Accuracy -  0.9219730941704036
Precision -  1.0
MNB
Accuracy -  0.9695067264573991
Precision -  1.0
DT
Accuracy -  0.9605381165919282
Precision -  0.9279279279279279
LR
Accuracy -  0.9775784753363229
Precision -  0.9453125
RF
Accuracy -  0.9865470852017937
Precision -  1.0
AdaBoost
Accuracy -  0.9820627802690582
Precision -  0.9612403100775194
BgC
Accuracy -  0.9802690582959641
Precision -  0.927007299270073
GBDT
Accuracy -  0.9730941704035875
Precision -  0.9739130434782609
xgb
Accuracy -  0.9856502242152466
Precision -  0.9767441860465116
```

```
y = clean_data["v1"].values
x = clean_data["messages"].values
```

```
from sklearn.model_selection import train_test_split
xtrain , xtest , ytrain , ytest = train_test_split(x,y, test_size = 0.2, random_state = 1)
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder

vectorizer = TfidfVectorizer()
X_train_vectorized = vectorizer.fit_transform(xtrain)
X_test_vectorized = vectorizer.transform(xtest)

label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(ytrain)
y_test_encoded = label_encoder.transform(ytest)

accuracy_scores2 = []
precision_scores2 = []

for name, clf in clfs.items():
    current_accuracy,current_precision  = train_classifier(clf, X_train_vectorized, y_train_encoded, X_test_vectorized, y_test_encoded)

    print(name)
    print("Accuracy - ", current_accuracy)
    print("Precision - ",current_precision)

    accuracy_scores.append(accuracy_scores2)
    accuracy_scores.append(precision_scores2)
```

```
SVC
Accuracy -  0.9856502242152466
Precision -  0.9694656488549618
KN
Accuracy -  0.9228699551569507
Precision -  1.0
MNB
Accuracy -  0.9730941704035875
Precision -  1.0
DT
Accuracy -  0.9201793721973094
Precision -  0.7314814814814815
LR
Accuracy -  0.9704035874439462
Precision -  0.9344262295081968
RF
Accuracy -  0.9829596412556054
Precision -  1.0
AdaBoost
Accuracy -  0.9668161434977578
Precision -  0.8923076923076924
BgC
Accuracy -  0.9605381165919282
Precision -  0.8625954198473282
GBDT
Accuracy -  0.947085201793722
Precision -  0.8846153846153846
```

```
accuracy_scores_unclean = accuracy_scores1
accuracy_scores_clean = accuracy_scores2

bar_width = 0.35

r1 = np.arange(len(clfs))
r2 = [x + bar_width for x in r1]

plt.figure(figsize=(12, 6))

plt.bar(r1, accuracy_scores_clean, color='b', width=bar_width, edgecolor='grey', label='Unclean Data')
plt.bar(r2, accuracy_scores_unclean, color='r', width=bar_width, edgecolor='grey', label='Cleaned Data')

plt.xlabel('Classifiers', fontweight='bold')
plt.ylabel('Accuracy', fontweight='bold')
plt.title('Comparison of Classifier Accuracy on Cleaned vs Uncleaned Data')
plt.xticks([r + bar_width / 2 for r in range(len(clfs))], clfs)

plt.legend()
plt.show()
```