https://colab.research.google.com/drive/1ReyPTUm9_YFte7dQiaSKaxE2mKK5fRy8#scrollTo=oIheeawyI5Pe
Naive Bayes

/////code////////

```
# -*- coding: utf-8 -*-
"""NaiveBayes.ipynb

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/1ReyPTUm9_YFte7dQiaSKaxE2mKK5fRy8
"""

import pandas as pd
import numpy as np
import re

test_data = pd.read_csv('test.csv')
df = pd.read_csv('train.csv')

print(test_data.shape)
print(df.shape)

fake_counts = test_data['fake'].value_counts()
print("Number of fake profiles:", fake_counts[1])
print("Number of real profiles:", fake_counts[0])

fake_count_training = df['fake'].value_counts()
print("Number of fake profiles:", fake_count_training[1])
print("Number of real profiles:", fake_count_training[0])

df.isnull().sum()

df.tail(5)

"""Features"""
```

```
df.columns

feature_cols = ['profile pic', 'nums/length username', 'fullname words',
       'nums/length fullname', 'name==username', 'description length',
       'external URL', 'private', '#posts', '#followers', '#follows']
X_train = df[feature_cols]

X_test = test_data[feature_cols]

y_test = test_data['fake']

y_train = df['fake']

y_train

"""#Model Training"""

# Commented out IPython magic to ensure Python compatibility.
import joblib
import matplotlib.pyplot as plt
# %matplotlib inline

from sklearn import impute
from sklearn import model_selection
from sklearn import metrics
from sklearn import preprocessing
from sklearn.metrics import roc_curve, auc, accuracy_score,
classification_report, confusion_matrix, accuracy_score, make_scorer
from sklearn.model_selection import StratifiedKFold, train_test_split,
GridSearchCV, learning_curve
from sklearn.naive_bayes import GaussianNB

def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
    """
    Generate a learning curve plot for a machine learning model.

    Parameters:
    - estimator: The machine learning model to evaluate.
    - title: The title of the plot.
```

```
    - X: The input features.
    - y: The target variable.
    - ylim: Tuple, optional, the y-axis limits.
    - cv: Cross-validation strategy. None by default.
    - n_jobs: Number of jobs to run in parallel for cross-validation.
    - train_sizes: Array of training set sizes.

    Returns:
    - Matplotlib plot object.
    """

    plt.figure()
    plt.title(title)

    if ylim is not None:
        plt.ylim(*ylim)

    plt.xlabel("Training examples")
    plt.ylabel("Score")

    # Generate learning curves using the learning_curve function
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)

    # Calculate mean and standard deviation of training scores
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)

    # Calculate mean and standard deviation of test scores
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.grid()
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1,
color="g")
```

```python
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt

def train(X_train,y_train,X_test):
    """ Trains and predicts dataset with a Gaussian Naive Bayes """

    gnb = GaussianNB()
    gnb.fit(X_train,y_train)
    print("The best classifier is: ",gnb)
    # Estimate score
    scores = model_selection.cross_val_score(gnb, X_train,y_train, cv=5)
    print (scores)
    print('Estimated score: %0.5f (+/- %0.5f)' % (scores.mean(),
scores.std() / 2))
    title = 'Learning Curves (Naive Bayes)'
    plot_learning_curve(gnb, title, X_train, y_train, cv=5)
    plt.show()
    # Predict
    y_pred = gnb.predict(X_test)
    return y_test,y_pred

# Start training models
y_test, y_pred = train(X_train, y_train, X_test)

"""Plot Confusion Matrix"""

def plot_confusion_matrix(cm, title='Confusion matrix Naive Bayes',
cmap=plt.cm.Blues):
    target_names=['Fake','Genuine(Real)']
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(target_names))
    plt.xticks(tick_marks, target_names)
    plt.yticks(tick_marks, target_names)
```

```
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

cm=confusion_matrix(y_test, y_pred)

print('Confusion Matrix[without Normalization]')
print(cm)

plot_confusion_matrix(cm)

print(classification_report(y_test, y_pred))
```

https://colab.research.google.com/drive/1ljaM969ZpETrPZw4B8K5sjYnWU85M3c8
Multi Layer Perceptron

```
# -*- coding: utf-8 -*-
"""Fake Instagram Profile Detection Model ....

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/1ljaM969ZpETrPZw4B8K5sjYnWU85M3c8
"""

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

import tensorflow as tf
```

```python
from tensorflow import keras
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import Accuracy

from sklearn import metrics
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import
classification_report,accuracy_score,roc_curve,confusion_matrix

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# Load the training dataset
instagram_df_train=pd.read_csv('train.csv')
instagram_df_train

# Load the testing data
instagram_df_test=pd.read_csv('test.csv')
instagram_df_test

"""# Statistical Analysis"""

instagram_df_train.head()

instagram_df_train.tail()

# Getting dataframe info
instagram_df_train.info()

# Get the statistical summary of the dataframe
instagram_df_train.describe()

# Checking if null values exist
instagram_df_train.isnull().sum()

# Get the number of unique values in the "profile pic" feature
instagram_df_train['profile pic'].value_counts()
```

```python
# Get the number of unique values in "fake" (Target column)
instagram_df_train['fake'].value_counts()

"""# Data Visualization"""

# Visualize the data
sns.countplot(instagram_df_train['fake'])
plt.show()

# Visualize the private column data
sns.countplot(instagram_df_train['private'])
plt.show()

# Visualize the "profile pic" column data
sns.countplot(instagram_df_train['profile pic'])
plt.show()

# Visualize the data
plt.figure(figsize = (20, 10))
sns.distplot(instagram_df_train['nums/length username'])
plt.show()

# Correlation plot
plt.figure(figsize=(20, 20))
cm = instagram_df_train.corr()
ax = plt.subplot()
sns.heatmap(cm, annot = True, ax = ax)
plt.show()

"""# Data Modelling"""

# Training and testing dataset (inputs)
X_train = instagram_df_train.drop(columns = ['fake'])
X_test = instagram_df_test.drop(columns = ['fake'])
X_train

# Training and testing dataset (Outputs)
y_train = instagram_df_train['fake']
y_test = instagram_df_test['fake']
```

```
y_train

# Scale the data before training the model
from sklearn.preprocessing import StandardScaler, MinMaxScaler

scaler_x = StandardScaler()
X_train = scaler_x.fit_transform(X_train)
X_test = scaler_x.transform(X_test)

y_train = tf.keras.utils.to_categorical(y_train, num_classes = 2)
y_test = tf.keras.utils.to_categorical(y_test, num_classes = 2)

y_train

import tensorflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

model = Sequential()
model.add(Dense(50, input_dim=11, activation='relu'))
model.add(Dense(150, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(150, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(25, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(2,activation='softmax'))

model.summary()

model.compile(optimizer = 'adam', loss = 'categorical_crossentropy',
metrics = ['accuracy'])

epochs_hist = model.fit(X_train, y_train, epochs = 50,  verbose = 1,
validation_split = 0.1)

"""# Model Validation and Results"""

print(epochs_hist.history.keys())
```

```
plt.plot(epochs_hist.history['loss'])
plt.plot(epochs_hist.history['val_loss'])

plt.title('Model Loss Progression During Training/Validation')
plt.ylabel('Training and Validation Losses')
plt.xlabel('Epoch Number')
plt.legend(['Training Loss', 'Validation Loss'])
plt.show()

predicted = model.predict(X_test)

predicted_value = []
test = []
for i in predicted:
    predicted_value.append(np.argmax(i))

for i in y_test:
    test.append(np.argmax(i))

print(classification_report(test, predicted_value))

plt.figure(figsize=(10, 10))
cm=confusion_matrix(test, predicted_value)
sns.heatmap(cm, annot=True)
plt.show()
```

https://colab.research.google.com/drive/16883jLs7yFhpHoWTgYYyLsOXfWfUTs0g?usp=sharing

Random Forest

```
# -*- coding: utf-8 -*-
"""RandomForest.ipynb

Automatically generated by Colab.
```

```
Original file is located at

https://colab.research.google.com/drive/16883jLs7yFhpHoWTgYYyLsOXfWfUTs0g
"""

import pandas as pd
import numpy as np
import re

test_data = pd.read_csv('test.csv')
df = pd.read_csv('train.csv')

print(test_data.shape)
print(df.shape)

df.isnull().sum()

df.tail(5)

"""Features"""

df.columns

feature_cols = ['profile pic', 'nums/length username', 'fullname words',
       'nums/length fullname', 'name==username', 'description length',
       'external URL', 'private', '#posts', '#followers', '#follows']
X_train = df[feature_cols]

X_test = test_data[feature_cols]

y_test = test_data['fake']



y_train = df['fake']

y_train

"""#Model Training"""
```

```python
# Commented out IPython magic to ensure Python compatibility.
import joblib
import matplotlib.pyplot as plt
# %matplotlib inline

from sklearn import impute
from sklearn import model_selection
from sklearn import metrics
from sklearn import preprocessing
from sklearn.metrics import roc_curve, auc, accuracy_score,
classification_report, confusion_matrix, accuracy_score, make_scorer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold, train_test_split,
GridSearchCV, learning_curve
from xgboost import XGBClassifier
from sklearn.svm import SVC

def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
    """
    Generate a learning curve plot for a machine learning model.

    Parameters:
    - estimator: The machine learning model to evaluate.
    - title: The title of the plot.
    - X: The input features.
    - y: The target variable.
    - ylim: Tuple, optional, the y-axis limits.
    - cv: Cross-validation strategy. None by default.
    - n_jobs: Number of jobs to run in parallel for cross-validation.
    - train_sizes: Array of training set sizes.

    Returns:
    - Matplotlib plot object.
    """

    plt.figure()
    plt.title(title)

    if ylim is not None:
```

```
        plt.ylim(*ylim)

    plt.xlabel("Training examples")
    plt.ylabel("Score")

    # Generate learning curves using the learning_curve function
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)

    # Calculate mean and standard deviation of training scores
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)

    # Calculate mean and standard deviation of test scores
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.grid()
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1,
color="g")

    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt

def train(X_train,y_train,X_test):
    """ Trains and predicts dataset with a Random Forest classifier """

    clf=RandomForestClassifier(n_estimators=40)
    clf.fit(X_train,y_train)
    print("The best classifier is: ",clf)
    # Estimate score
```

```
    scores = model_selection.cross_val_score(clf, X_train,y_train, cv=5)
    print (scores)
    print('Estimated score: %0.5f (+/- %0.5f)' % (scores.mean(),
scores.std() / 2))
    title = 'Learning Curves (Random Forest)'
    plot_learning_curve(clf, title, X_train, y_train, cv=5)
    plt.show()
    # Predict
    y_pred = clf.predict(X_test)
    # print("output")
    # print(clf.predict([[100,872,924,20,15,2,3]]))
    # print(clf.predict([[1234,15,104,1150,0,2,1]]))
    # print(clf.predict([[60,14,592,0,0,-2,1]]))
    # print(clf.predict([[16935,2242,918,44,124,-2,1]]))
    # print(clf.predict([[1294,254,276,89,8,2,5]]))
    # print(clf.predict([[44,21,614,0,0,-2,1]]))#fake
    # print(clf.predict([[1386,88,267,106,0,2,1]]))#g
    # print(clf.predict([[4149,279,64,20,22,-1,1]]))#g
    # print(clf.predict([[732,191,656,1,1,1,3]]))#g
    return y_test,y_pred

# Start training models
y_test, y_pred = train(X_train, y_train, X_test)

"""Plot Confusion Matrix"""

def plot_confusion_matrix(cm, title='Confusion matrix',
cmap=plt.cm.Blues):
    target_names=['Fake','Genuine(Real)']
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(target_names))
    plt.xticks(tick_marks, target_names)
    plt.yticks(tick_marks, target_names)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

cm=confusion_matrix(y_test, y_pred)
```

```
print('Confusion Matrix[without Normalization]')
print(cm)

plot_confusion_matrix(cm)

print(classification_report(y_test,y_pred))
```

https://colab.research.google.com/drive/1Yv9oblxdbrsah0nmZriM4YCW7LRbjsSP?usp=sharing
SVM

```
# -*- coding: utf-8 -*-
"""SVM.ipynb

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/1Yv9oblxdbrsah0nmZriM4YCW7LRbjsSP
"""

import pandas as pd
import numpy as np
import re

test_data = pd.read_csv('test.csv')
df = pd.read_csv('train.csv')

print(test_data.shape)
print(df.shape)

df.isnull().sum()
```

```
df.tail(5)

"""Features"""

df.columns

feature_cols = ['profile pic', 'nums/length username', 'fullname words',
       'nums/length fullname', 'name==username', 'description length',
       'external URL', 'private', '#posts', '#followers', '#follows']
X_train = df[feature_cols]

X_test = test_data[feature_cols]

y_test = test_data['fake']



y_train = df['fake']

y_train

"""#Model Training"""

# Commented out IPython magic to ensure Python compatibility.
import joblib
import matplotlib.pyplot as plt
# %matplotlib inline

from sklearn import impute
from sklearn import model_selection
from sklearn import metrics
from sklearn import preprocessing
from sklearn.metrics import roc_curve, auc, accuracy_score,
classification_report, confusion_matrix, accuracy_score, make_scorer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold, train_test_split,
GridSearchCV, learning_curve
from xgboost import XGBClassifier
from sklearn.svm import SVC
```

```python
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
    """
    Generate a learning curve plot for a machine learning model.

    Parameters:
    - estimator: The machine learning model to evaluate.
    - title: The title of the plot.
    - X: The input features.
    - y: The target variable.
    - ylim: Tuple, optional, the y-axis limits.
    - cv: Cross-validation strategy. None by default.
    - n_jobs: Number of jobs to run in parallel for cross-validation.
    - train_sizes: Array of training set sizes.

    Returns:
    - Matplotlib plot object.
    """

    plt.figure()
    plt.title(title)

    if ylim is not None:
        plt.ylim(*ylim)

    plt.xlabel("Training examples")
    plt.ylabel("Score")

    # Generate learning curves using the learning_curve function
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)

    # Calculate mean and standard deviation of training scores
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)

    # Calculate mean and standard deviation of test scores
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
```

```
    plt.grid()
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1,
color="g")

    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt

def train(X_train, y_train, X_test, y_test):
    filename = 'model.ckpt'
    param_grid = {
        'C': [0.1, 1, 10],
        'gamma': [0.01, 0.1, 1],
        'kernel': ['rbf']
    }

    svm_clf = SVC()
    scorer = make_scorer(accuracy_score)
    grid_search = GridSearchCV(svm_clf, param_grid, cv=5, scoring=scorer,
n_jobs=-1)
    grid_search.fit(X_train, y_train)

    best_svm_model = grid_search.best_estimator_
    joblib.dump(best_svm_model, filename)

    print("Training Datasets using SVC")
    print("Best Hyperparameters:", grid_search.best_params_)
    y_pred = best_svm_model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print("Accuracy of the Best Model:", accuracy)

    title = 'Learning Curves (SVM)'
```

```
    9
    plot_learning_curve(best_svm_model, title, X_train, y_train, cv=5)
    plt.show()
    # print("output svm ")
    # print(best_svm_model.predict([[-2,16935,2242,918,44,124,1]]))#real
    # print(best_svm_model.predict([[2,1234,15,104,1150,0,1]]))#fake
    # print(best_svm_model.predict([[-2,60,14,592,0,0,1]]))#fake
    # print(best_svm_model.predict([[2,1294,254,276,89,8,5]]))#real
    # print(best_svm_model.predict([[2,20370,5470,2385,145,52,5]]))#real
    # print(best_svm_model.predict([[-2,3131,506,381,9,40,1]]))#real
    # print(best_svm_model.predict([[0,4024,264,87,323,16,1]]))#real
    # print(best_svm_model.predict([[0,40586,640,622,1118,32,1]]))#real
    # print(best_svm_model.predict([[2,2016,62,64,13,0,5]]))#real
    # print(best_svm_model.predict([[2,2016,62,64,13,0,5]]))#real

    return y_test, y_pred

# Start training models
y_test, y_pred = train(X_train, y_train, X_test, y_test)

"""Plot Confusion Matrix"""

def plot_confusion_matrix(cm, title='Confusion matrix',
cmap=plt.cm.Blues):
    target_names=['Fake','Genuine(Real)']
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(target_names))
    plt.xticks(tick_marks, target_names)
    plt.yticks(tick_marks, target_names)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

cm=confusion_matrix(y_test, y_pred)

print('Confusion Matrix[without Normalization]')
print(cm)
```

```
plot_confusion_matrix(cm)

print(classification_report(y_test,y_pred))
```

https://colab.research.google.com/drive/1jC1Q7aSg5ctlvhFZivAkwgwhACuzgDGE?usp=sharing
KNN

```
# -*- coding: utf-8 -*-
"""KNN.ipynb

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/1jC1Q7aSg5ctlvhFZivAkwgwhACuzgDGE
"""

import pandas as pd
import numpy as np
import re

test_data = pd.read_csv('test.csv')
df = pd.read_csv('train.csv')

print(test_data.shape)
print(df.shape)

df.isnull().sum()

df.tail(5)

"""Features"""
```

```python
df.columns

feature_cols = ['profile pic', 'nums/length username', 'fullname words',
       'nums/length fullname', 'name==username', 'description length',
       'external URL', 'private', '#posts', '#followers', '#follows']
X_train = df[feature_cols]

X_test = test_data[feature_cols]

y_test = test_data['fake']

y_train = df['fake']

y_train

"""#Model Training"""

# Commented out IPython magic to ensure Python compatibility.
import joblib
import matplotlib.pyplot as plt
# %matplotlib inline

from sklearn import impute
from sklearn import model_selection
from sklearn import metrics
from sklearn import preprocessing
from sklearn.metrics import roc_curve, auc, accuracy_score,
classification_report, confusion_matrix, accuracy_score, make_scorer
from sklearn.model_selection import StratifiedKFold, train_test_split,
GridSearchCV, learning_curve
from sklearn.neighbors import KNeighborsClassifier

def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
    """
    Generate a learning curve plot for a machine learning model.

    Parameters:
    - estimator: The machine learning model to evaluate.
```

```
    - title: The title of the plot.
    - X: The input features.
    - y: The target variable.
    - ylim: Tuple, optional, the y-axis limits.
    - cv: Cross-validation strategy. None by default.
    - n_jobs: Number of jobs to run in parallel for cross-validation.
    - train_sizes: Array of training set sizes.

    Returns:
    - Matplotlib plot object.
    """

    plt.figure()
    plt.title(title)

    if ylim is not None:
        plt.ylim(*ylim)

    plt.xlabel("Training examples")
    plt.ylabel("Score")

    # Generate learning curves using the learning_curve function
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)

    # Calculate mean and standard deviation of training scores
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)

    # Calculate mean and standard deviation of test scores
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.grid()
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1,
color="g")
```

```python
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt

def train(X_train,y_train,X_test):
    """ Trains and predicts dataset with a K Neighbours """

    knn = KNeighborsClassifier(n_neighbors=3)
    knn.fit(X_train,y_train)
    print("The best classifier is: ",knn)
    # Estimate score
    scores = model_selection.cross_val_score(knn, X_train,y_train, cv=5)
    print (scores)
    print('Estimated score: %0.5f (+/- %0.5f)' % (scores.mean(),
scores.std() / 2))
    title = 'Learning Curves (K Neighbours)'
    plot_learning_curve(knn, title, X_train, y_train, cv=5)
    plt.show()
    # Predict
    y_pred = knn.predict(X_test)
    # print("output")
    # print(clf.predict([[100,872,924,20,15,2,3]]))
    # print(clf.predict([[1234,15,104,1150,0,2,1]]))
    # print(clf.predict([[60,14,592,0,0,-2,1]]))
    # print(clf.predict([[16935,2242,918,44,124,-2,1]]))
    # print(clf.predict([[1294,254,276,89,8,2,5]]))
    # print(clf.predict([[44,21,614,0,0,-2,1]]))#fake
    # print(clf.predict([[1386,88,267,106,0,2,1]]))#g
    # print(clf.predict([[4149,279,64,20,22,-1,1]]))#g
    # print(clf.predict([[732,191,656,1,1,1,3]]))#g
    return y_test,y_pred

# Start training models
y_test, y_pred = train(X_train, y_train, X_test)
```

```
"""Plot Confusion Matrix"""

def plot_confusion_matrix(cm, title='Confusion matrix',
cmap=plt.cm.Blues):
    target_names=['Fake','Genuine(Real)']
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(target_names))
    plt.xticks(tick_marks, target_names)
    plt.yticks(tick_marks, target_names)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

cm=confusion_matrix(y_test, y_pred)

print('Confusion Matrix[without Normalization]')
print(cm)

plot_confusion_matrix(cm)

print(classification_report(y_test,y_pred))
```

https://colab.research.google.com/drive/16zIT7IjM-SguoQFEVgRabOP7G4LInRD7?usp=sharing

Logistic Regression

```
# -*- coding: utf-8 -*-
"""LogisticRegression.ipynb
```

```
Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/16zIT7IjM-SguoQFEVgRabOP7G4LlnRD7
"""

import pandas as pd
import numpy as np
import re

test_data = pd.read_csv('test.csv')
df = pd.read_csv('train.csv')

print(test_data.shape)
print(df.shape)

df.isnull().sum()

df.tail(5)

"""Features"""

df.columns

feature_cols = ['profile pic', 'nums/length username', 'fullname words',
       'nums/length fullname', 'name==username', 'description length',
       'external URL', 'private', '#posts', '#followers', '#follows']
X_train = df[feature_cols]

X_test = test_data[feature_cols]

y_test = test_data['fake']

y_train = df['fake']

y_train

"""#Model Training"""
```

```python
# Commented out IPython magic to ensure Python compatibility.
import joblib
import matplotlib.pyplot as plt
# %matplotlib inline

from sklearn import impute
from sklearn import model_selection
from sklearn import metrics
from sklearn import preprocessing
from sklearn.metrics import roc_curve, auc, accuracy_score,
classification_report, confusion_matrix, accuracy_score, make_scorer
from sklearn.model_selection import StratifiedKFold, train_test_split,
GridSearchCV, learning_curve
from sklearn.linear_model import LogisticRegression

def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
    """
    Generate a learning curve plot for a machine learning model.

    Parameters:
    - estimator: The machine learning model to evaluate.
    - title: The title of the plot.
    - X: The input features.
    - y: The target variable.
    - ylim: Tuple, optional, the y-axis limits.
    - cv: Cross-validation strategy. None by default.
    - n_jobs: Number of jobs to run in parallel for cross-validation.
    - train_sizes: Array of training set sizes.

    Returns:
    - Matplotlib plot object.
    """

    plt.figure()
    plt.title(title)

    if ylim is not None:
        plt.ylim(*ylim)
```

```
    plt.xlabel("Training examples")
    plt.ylabel("Score")

    # Generate learning curves using the learning_curve function
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)

    # Calculate mean and standard deviation of training scores
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)

    # Calculate mean and standard deviation of test scores
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.grid()
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1,
color="g")

    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt

def train(X_train,y_train,X_test):
    """ Trains and predicts dataset with a Logistic Regression """

    clf = LogisticRegression(random_state=0,max_iter=500)
    clf.fit(X_train,y_train)
    print("The best classifier is: ",clf)
    # Estimate score
    scores = model_selection.cross_val_score(clf, X_train,y_train, cv=5)
    print (scores)
```

```
    print('Estimated score: %0.5f (+/- %0.5f)' % (scores.mean(),
scores.std() / 2))
    title = 'Learning Curves (Logistic Regression)'
    plot_learning_curve(clf, title, X_train, y_train, cv=5)
    plt.show()
    # Predict
    y_pred = clf.predict(X_test)
    # print("output")
    # print(clf.predict([[100,872,924,20,15,2,3]]))
    # print(clf.predict([[1234,15,104,1150,0,2,1]]))
    # print(clf.predict([[60,14,592,0,0,-2,1]]))
    # print(clf.predict([[16935,2242,918,44,124,-2,1]]))
    # print(clf.predict([[1294,254,276,89,8,2,5]]))
    # print(clf.predict([[44,21,614,0,0,-2,1]]))#fake
    # print(clf.predict([[1386,88,267,106,0,2,1]]))#g
    # print(clf.predict([[4149,279,64,20,22,-1,1]]))#g
    # print(clf.predict([[732,191,656,1,1,1,3]]))#g
    return y_test,y_pred

# Start training models
y_test, y_pred = train(X_train, y_train, X_test)

"""Plot Confusion Matrix"""

def plot_confusion_matrix(cm, title='Confusion matrix',
cmap=plt.cm.Blues):
    target_names=['Fake','Genuine(Real)']
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(target_names))
    plt.xticks(tick_marks, target_names)
    plt.yticks(tick_marks, target_names)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

cm=confusion_matrix(y_test, y_pred)

print('Confusion Matrix[without Normalization]')
```

```
print(cm)

plot_confusion_matrix(cm)

print(classification_report(y_test,y_pred))
```

https://colab.research.google.com/drive/1AE1qmi0ERF5DUnJfkr-POHi8T5svRBbF?usp=sharing
Gradient Boost Classifier

```
# -*- coding: utf-8 -*-
"""GradientBoost.ipynb

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/1AE1qmi0ERF5DUnJfkr-POHi8T5svRBbF
"""

import pandas as pd
import numpy as np
import re

test_data = pd.read_csv('test.csv')
df = pd.read_csv('train.csv')

print(test_data.shape)
print(df.shape)

df.isnull().sum()

df.tail(5)

"""Features"""
```

```
df.columns

feature_cols = ['profile pic', 'nums/length username', 'fullname words',
       'nums/length fullname', 'name==username', 'description length',
       'external URL', 'private', '#posts', '#followers', '#follows']
X_train = df[feature_cols]

X_test = test_data[feature_cols]

y_test = test_data['fake']

y_train = df['fake']

y_train

"""#Model Training"""

# Commented out IPython magic to ensure Python compatibility.
import joblib
import matplotlib.pyplot as plt
# %matplotlib inline

from sklearn import impute
from sklearn import model_selection
from sklearn import metrics
from sklearn import preprocessing
from sklearn.metrics import roc_curve, auc, accuracy_score,
classification_report, confusion_matrix, accuracy_score, make_scorer
from sklearn.model_selection import StratifiedKFold, train_test_split,
GridSearchCV, learning_curve
from sklearn.ensemble import GradientBoostingClassifier

def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
    """
    Generate a learning curve plot for a machine learning model.

    Parameters:
    - estimator: The machine learning model to evaluate.
```

```
    - title: The title of the plot.
    - X: The input features.
    - y: The target variable.
    - ylim: Tuple, optional, the y-axis limits.
    - cv: Cross-validation strategy. None by default.
    - n_jobs: Number of jobs to run in parallel for cross-validation.
    - train_sizes: Array of training set sizes.

    Returns:
    - Matplotlib plot object.
    """

    plt.figure()
    plt.title(title)

    if ylim is not None:
        plt.ylim(*ylim)

    plt.xlabel("Training examples")
    plt.ylabel("Score")

    # Generate learning curves using the learning_curve function
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)

    # Calculate mean and standard deviation of training scores
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)

    # Calculate mean and standard deviation of test scores
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.grid()
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1,
color="g")
```

```
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt

def train(X_train,y_train,X_test):
    """ Trains and predicts dataset with a Gradient Boost """

    clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
max_depth=1, random_state=0)
    clf.fit(X_train,y_train)
    print("The best classifier is: ",clf)
    # Estimate score
    scores = model_selection.cross_val_score(clf, X_train,y_train, cv=5)
    print (scores)
    print('Estimated score: %0.5f (+/- %0.5f)' % (scores.mean(),
scores.std() / 2))
    title = 'Learning Curves (Gradient Boost)'
    plot_learning_curve(clf, title, X_train, y_train, cv=5)
    plt.show()
    # Predict
    y_pred = clf.predict(X_test)
    return y_test,y_pred

# Start training models
y_test, y_pred = train(X_train, y_train, X_test)

"""Plot Confusion Matrix"""

def plot_confusion_matrix(cm, title='Confusion matrix',
cmap=plt.cm.Blues):
    target_names=['Fake','Genuine(Real)']
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(target_names))
```

```
    plt.xticks(tick_marks, target_names)
    plt.yticks(tick_marks, target_names)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

cm=confusion_matrix(y_test, y_pred)

print('Confusion Matrix[without Normalization]')
print(cm)

plot_confusion_matrix(cm)

print(classification_report(y_test,y_pred))
```