TikTok Classifying videos using machine learning

January 17, 2025

1 TikTok Project

Course 6 - The Nuts and bolts of machine learning

Recall that you are a data professional at TikTok. Your supervisor was impressed with the work you have done and has requested that you build a machine learning model that can be used to determine whether a video contains a claim or whether it offers an opinion. With a successful prediction model, TikTok can reduce the backlog of user reports and prioritize them more efficiently.

A notebook was structured and prepared to help you in this project. A notebook was structured and prepared to help you in this project. Please complete the following questions.

2 Course 6 End-of-course project: Classifying videos using machine learning

In this activity, you will practice using machine learning techniques to predict on a binary outcome variable.

The purpose of this model is to increase response time and system efficiency by automating the initial stages of the claims process.

The goal of this model is to predict whether a TikTok video presents a "claim" or presents an "opinion".

This activity has three parts:

Part 1: Ethical considerations * Consider the ethical implications of the request

- Should the objective of the model be adjusted?

Part 2: Feature engineering

- Perform feature selection, extraction, and transformation to prepare the data for modeling

Part 3: Modeling

- Build the models, evaluate them, and advise on next steps

Follow the instructions and answer the questions below to complete the activity. Then, you will complete an Executive Summary using the questions listed on the PACE Strategy Document.

Be sure to complete this activity before moving on. The next course item will provide you with a completed exemplar to compare to your own work.

3 Classify videos using machine learning

4 PACE stages

Throughout these project notebooks, you'll see references to the problem-solving framework PACE. The following notebook components are labeled with the respective PACE stage: Plan, Analyze, Construct, and Execute.

4.1 PACE: Plan

Consider the questions in your PACE Strategy Document to reflect on the Plan stage.

In this stage, consider the following questions:

1. **What are you being asked to do? What metric should I use to evaluate success of my business/organizational objective?**
2. **What are the ethical implications of the model? What are the consequences of your model making errors?**

- What is the likely effect of the model when it predicts a false negative (i.e., when the model says a video does not contain a claim and it actually does)?
- What is the likely effect of the model when it predicts a false positive (i.e., when the model says a video does contain a claim and it actually does not)?

3. **How would you proceed?**

Answers:

1. Build a model to classify TikTok videos as claims or opinions, helping moderators quickly identify potential violations.
2. Ethical implications Minimize false negatives (when a claim is classified as an opinion) to ensure rule-breaking videos are not missed. Evaluation Metric: Recall.
3. Steps
    - Split the data into training, validation, and test sets (60/20/20).
    - Train models and tune hyperparameters on the training set.
    - Select the best model on the validation set.
    - Assess the performance of the chosen model on the test set.

4.1.1 Task 1. Imports and data loading

Start by importing packages needed to build machine learning models to achieve the goal of this project.

[1]:
```
# Import packages for data manipulation
### YOUR CODE HERE ###
import pandas as pd
import numpy as np
```

```
# Import packages for data visualization
### YOUR CODE HERE ###
import matplotlib.pyplot as plt
import seaborn as sns

# Import packages for data preprocessing
### YOUR CODE HERE ###
from sklearn.feature_extraction.text import CountVectorizer


# Import packages for data modeling
### YOUR CODE HERE ###
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪precision_score, \
recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay

from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from xgboost import plot_importance
```

Now load the data from the provided csv file into a dataframe.

Note: As shown in this cell, the dataset has been automatically loaded in for you. You do not need to download the .csv file, or provide more code, in order to access the dataset and proceed with this lab. Please continue with this activity by completing the following instructions.

[2]:
```
# Load dataset into dataframe
data = pd.read_csv("tiktok_dataset.csv")
```

4.2 PACE: Analyze

Consider the questions in your PACE Strategy Document to reflect on the Analyze stage.

4.2.1 Task 2: Examine data, summary info, and descriptive stats

Inspect the first five rows of the dataframe.

[3]:
```
# Display first few rows
### YOUR CODE HERE ###
data.head(5)
```

[3]:

	# claim_status	video_id	video_duration_sec	\
0	1	claim	7017666017	59
1	2	claim	4014381136	32
2	3	claim	9859838091	31
3	4	claim	1866847991	25
4	5	claim	7105231098	19

```
                              video_transcription_text verified_status  \
0  someone shared with me that drone deliveries a…    not verified
1  someone shared with me that there are more mic…    not verified
2  someone shared with me that american industria…    not verified
3  someone shared with me that the metro of st. p…    not verified
4  someone shared with me that the number of busi…    not verified

  author_ban_status  video_view_count  video_like_count  video_share_count  \
0      under review          343296.0           19425.0              241.0
1            active          140877.0           77355.0            19034.0
2            active          902185.0           97690.0             2858.0
3            active          437506.0          239954.0            34812.0
4            active           56167.0           34987.0             4110.0

   video_download_count  video_comment_count
0                   1.0                  0.0
1                1161.0                684.0
2                 833.0                329.0
3                1234.0                584.0
4                 547.0                152.0
```

Get the number of rows and columns in the dataset.

[4]:
```
# Get number of rows and columns
### YOUR CODE HERE ###
data.shape
```

[4]:
```
(19382, 12)
```

Get the data types of the columns.

[5]:
```
# Get data types of columns
### YOUR CODE HERE ###
data.dtypes
```

[5]:
```
#                             int64
claim_status                 object
video_id                      int64
video_duration_sec            int64
video_transcription_text     object
verified_status              object
author_ban_status            object
video_view_count            float64
video_like_count            float64
video_share_count           float64
video_download_count        float64
video_comment_count         float64
```

```
dtype: object
```

Get basic information about the dataset.

[6]:
```
# Get basic information
### YOUR CODE HERE ###
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19382 entries, 0 to 19381
Data columns (total 12 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   #                         19382 non-null  int64
 1   claim_status              19084 non-null  object
 2   video_id                  19382 non-null  int64
 3   video_duration_sec        19382 non-null  int64
 4   video_transcription_text  19084 non-null  object
 5   verified_status           19382 non-null  object
 6   author_ban_status         19382 non-null  object
 7   video_view_count          19084 non-null  float64
 8   video_like_count          19084 non-null  float64
 9   video_share_count         19084 non-null  float64
 10  video_download_count      19084 non-null  float64
 11  video_comment_count       19084 non-null  float64
dtypes: float64(5), int64(3), object(4)
memory usage: 1.8+ MB
```

Generate basic descriptive statistics about the dataset.

[7]:
```
# Generate basic descriptive stats
### YOUR CODE HERE ###
data.describe()
```

[7]:

	#	video_id	video_duration_sec	video_view_count	\
count	19382.000000	1.938200e+04	19382.000000	19084.000000	
mean	9691.500000	5.627454e+09	32.421732	254708.558688	
std	5595.245794	2.536440e+09	16.229967	322893.280814	
min	1.000000	1.234959e+09	5.000000	20.000000	
25%	4846.250000	3.430417e+09	18.000000	4942.500000	
50%	9691.500000	5.618664e+09	32.000000	9954.500000	
75%	14536.750000	7.843960e+09	47.000000	504327.000000	
max	19382.000000	9.999873e+09	60.000000	999817.000000	

	video_like_count	video_share_count	video_download_count	\
count	19084.000000	19084.000000	19084.000000	
mean	84304.636030	16735.248323	1049.429627	
std	133420.546814	32036.174350	2004.299894	

```
min            0.000000          0.000000             0.000000
25%          810.750000        115.000000             7.000000
50%         3403.500000        717.000000            46.000000
75%       125020.000000      18222.000000          1156.250000
max       657830.000000     256130.000000         14994.000000

       video_comment_count
count         19084.000000
mean            349.312146
std             799.638865
min               0.000000
25%               1.000000
50%               9.000000
75%             292.000000
max            9599.000000
```

Check for and handle missing values.

[8]:
```
# Check for missing values
### YOUR CODE HERE ###
data.isna().sum()
```

[8]:
```
#                           0
claim_status                298
video_id                      0
video_duration_sec            0
video_transcription_text    298
verified_status               0
author_ban_status             0
video_view_count            298
video_like_count            298
video_share_count           298
video_download_count        298
video_comment_count         298
dtype: int64
```

[9]:
```
# Drop rows with missing values
### YOUR CODE HERE ###
data = data.dropna(axis=0)
```

[10]:
```
# Display first few rows after handling missing values
### YOUR CODE HERE ###
data.head(5)
```

[10]:
```
   # claim_status    video_id  video_duration_sec  \
0  1        claim  7017666017                  59
1  2        claim  4014381136                  32
```

```
2  3        claim  9859838091                 31
3  4        claim  1866847991                 25
4  5        claim  7105231098                 19

                           video_transcription_text verified_status  \
0  someone shared with me that drone deliveries a…    not verified
1  someone shared with me that there are more mic…    not verified
2  someone shared with me that american industria…    not verified
3  someone shared with me that the metro of st. p…    not verified
4  someone shared with me that the number of busi…    not verified

  author_ban_status  video_view_count  video_like_count  video_share_count  \
0      under review          343296.0           19425.0              241.0
1            active          140877.0           77355.0            19034.0
2            active          902185.0           97690.0             2858.0
3            active          437506.0          239954.0            34812.0
4            active           56167.0           34987.0             4110.0

   video_download_count  video_comment_count
0                   1.0                  0.0
1                1161.0                684.0
2                 833.0                329.0
3                1234.0                584.0
4                 547.0                152.0
```

Check for and handle duplicates.

[11]:
```
# Check for duplicates
### YOUR CODE HERE ###
data.duplicated().sum()
```

[11]:
```
0
```

Check for and handle outliers.

Answer: Tree models do not require handling of outliers

Check class balance.

[12]:
```
# Check class balance
### YOUR CODE HERE ###
data["claim_status"].value_counts(normalize=True)
```

[12]:
```
claim_status
claim      0.503458
opinion    0.496542
Name: proportion, dtype: float64
```

4.3 PACE: Construct

Consider the questions in your PACE Strategy Document to reflect on the Construct stage.

4.3.1 Task 3: Feature engineering

Extract the length of each `video_transcription_text` and add this as a column to the dataframe, so that it can be used as a potential feature in the model.

[13]:
```
# Extract the length of each `video_transcription_text` and add this as a␣
↪column to the dataframe
### YOUR CODE HERE ###
data['text_length'] = data['video_transcription_text'].str.len()
data.head(5)
```

[13]:

	#	claim_status	video_id	video_duration_sec	video_transcription_text	verified_status	author_ban_status	video_view_count	video_like_count	video_share_count	video_download_count	video_comment_count	text_length
0	1	claim	7017666017	59	someone shared with me that drone deliveries a…	not verified	under review	343296.0	19425.0	241.0	1.0	0.0	97
1	2	claim	4014381136	32	someone shared with me that there are more mic…	not verified	active	140877.0	77355.0	19034.0	1161.0	684.0	107
2	3	claim	9859838091	31	someone shared with me that american industria…	not verified	active	902185.0	97690.0	2858.0	833.0	329.0	137
3	4	claim	1866847991	25	someone shared with me that the metro of st. p…	not verified	active	437506.0	239954.0	34812.0	1234.0	584.0	131
4	5	claim	7105231098	19	someone shared with me that the number of busi…	not verified	active	56167.0	34987.0	4110.0	547.0	152.0	128

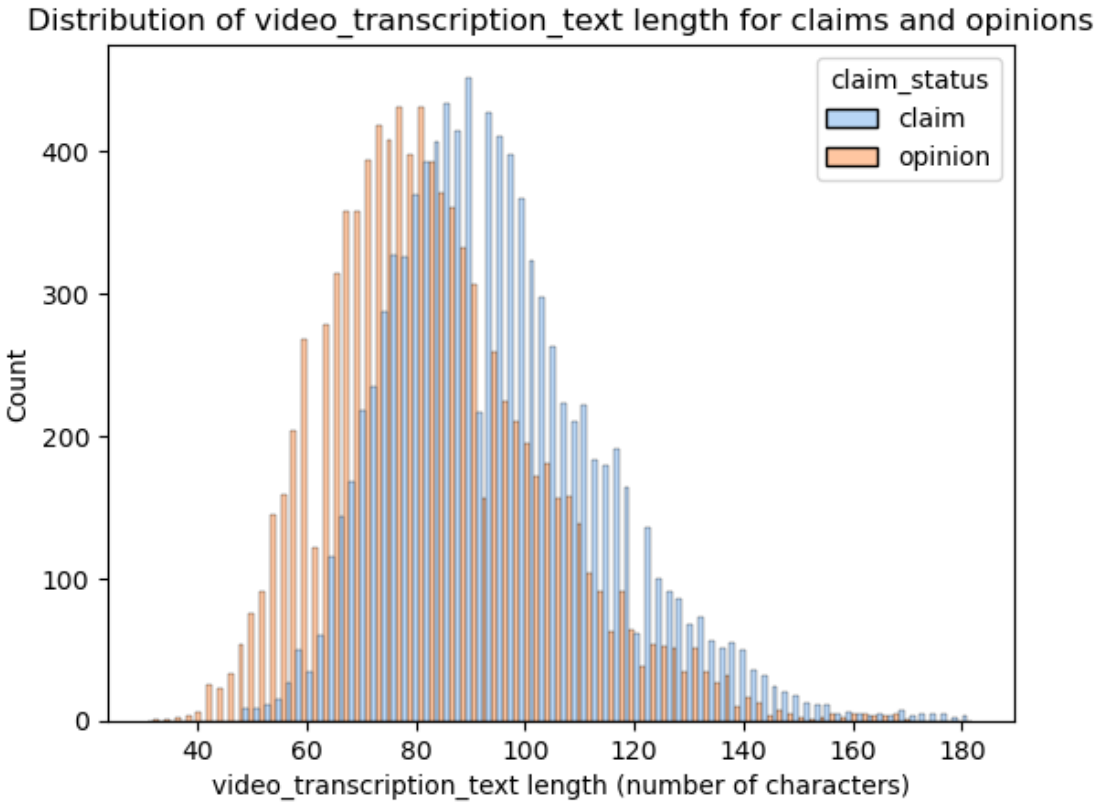
Calculate the average text_length for claims and opinions.

[14]:
```
# Calculate the average text_length for claims and opinions
### YOUR CODE HERE ###
data[['claim_status', 'text_length']].groupby('claim_status').mean()
```

[14]:
```
                text_length
claim_status
claim             95.376978
opinion           82.722562
```

Visualize the distribution of `text_length` for claims and opinions.

[15]:
```
# Visualize the distribution of `text_length` for claims and opinions
# Create two histograms in one plot
### YOUR CODE HERE ###
sns.histplot(data=data, stat="count", multiple="dodge", x="text_length",
             kde=False, palette="pastel", hue="claim_status",
             element="bars", legend=True)
plt.xlabel("video_transcription_text length (number of characters)")
plt.ylabel("Count")
plt.title("Distribution of video_transcription_text length for claims and␣
  ↪opinions")
plt.show()
```



Feature selection and transformation

Encode target and catgorical variables.

[16]:
```
# Create a copy of the X data
### YOUR CODE HERE ###
X = data.copy()

# Drop unnecessary columns
### YOUR CODE HERE ###
X = X.drop(['#', 'video_id', 'video_transcription_text'], axis=1)

# Encode target variable
### YOUR CODE HERE ###
X['claim_status'] = X['claim_status'].replace({'opinion': 0, 'claim': 1})

# Dummy encode remaining categorical values
### YOUR CODE HERE ###
X = pd.get_dummies(X, columns=['verified_status', 'author_ban_status'],␣
 ↪drop_first=True)

X.head()
```

[16]:

	claim_status	video_duration_sec	video_view_count	video_like_count	\
0	1	59	343296.0	19425.0	
1	1	32	140877.0	77355.0	
2	1	31	902185.0	97690.0	
3	1	25	437506.0	239954.0	
4	1	19	56167.0	34987.0	

	video_share_count	video_download_count	video_comment_count	text_length	\
0	241.0	1.0	0.0	97	
1	19034.0	1161.0	684.0	107	
2	2858.0	833.0	329.0	137	
3	34812.0	1234.0	584.0	131	
4	4110.0	547.0	152.0	128	

	verified_status_verified	author_ban_status_banned	\
0	False	False	
1	False	False	
2	False	False	
3	False	False	
4	False	False	

	author_ban_status_under review
0	True
1	False
2	False
3	False
4	False

4.3.2 Task 4: Split the data

Assign target variable.

[17]:
```
# Isolate target variable
### YOUR CODE HERE ###
y = X['claim_status']
```

Isolate the features.

[18]:
```
# Isolate features
### YOUR CODE HERE ###
X = X.drop(['claim_status'], axis=1)

# Display first few rows of features dataframe
### YOUR CODE HERE ###
X.head(5)
```

[18]:

	video_duration_sec	video_view_count	video_like_count	video_share_count	video_download_count	video_comment_count	text_length	verified_status_verified	author_ban_status_banned	author_ban_status_under review
0	59	343296.0	19425.0	241.0	1.0	0.0	97	False	False	True
1	32	140877.0	77355.0	19034.0	1161.0	684.0	107	False	False	False
2	31	902185.0	97690.0	2858.0	833.0	329.0	137	False	False	False
3	25	437506.0	239954.0	34812.0	1234.0	584.0	131	False	False	False
4	19	56167.0	34987.0	4110.0	547.0	152.0	128	False	False	False

4.3.3 Task 5: Create train/validate/test sets

Split data into training and testing sets, 80/20.

[19]:
```
# Split the data into training and testing sets
### YOUR CODE HERE ###
X_tr, X_test, y_tr, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=0)
```

Split the training set into training and validation sets, 75/25, to result in a final ratio of 60/20/20 for train/validate/test sets.

[20]:
```
# Split the training data into training and validation sets
### YOUR CODE HERE ###
X_train, X_val, y_train, y_val = train_test_split(X_tr, y_tr, test_size=0.25,␣
 ↪random_state=0)
```

Confirm that the dimensions of the training, validation, and testing sets are in alignment.

[21]:
```
# Get shape of each training, validation, and testing set
### YOUR CODE HERE ###
X_train.shape, X_val.shape, X_test.shape, y_train.shape, y_val.shape, y_test.
 ↪shape
```

[21]:
```
((11450, 10), (3817, 10), (3817, 10), (11450,), (3817,), (3817,))
```

4.3.4 Task 6. Build models

4.3.5 Build a random forest model

Fit a random forest model to the training set. Use cross-validation to tune the hyperparameters and select the model that performs best on recall.

[22]:
```
# Instantiate the random forest classifier
### YOUR CODE HERE ###
rf = RandomForestClassifier(random_state = 0)

# Create a dictionary of hyperparameters to tune
### YOUR CODE HERE ###
cv_params = {'max_depth': [5, 7, None],
             'max_features': [0.3, 0.6],
             'max_samples': [0.7],
             'min_samples_leaf': [1,2],
             'min_samples_split': [2,3],
             'n_estimators': [75,100,200],
             }

# Define a dictionary of scoring metrics to capture
### YOUR CODE HERE ###
scoring = {'accuracy', 'precision', 'recall', 'f1'}

# Instantiate the GridSearchCV object
```

```
### YOUR CODE HERE ###
rf_cv = GridSearchCV(rf, cv_params, scoring=scoring, cv=5, refit='recall')
```

[23]:
```
%%time
# Fit model with training set
rf_cv.fit(X_train, y_train)
```

```
CPU times: user 4min 58s, sys: 496 ms, total: 4min 58s
Wall time: 4min 58s
```

[23]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=0),
             param_grid={'max_depth': [5, 7, None], 'max_features': [0.3, 0.6],
                         'max_samples': [0.7], 'min_samples_leaf': [1, 2],
                         'min_samples_split': [2, 3],
                         'n_estimators': [75, 100, 200]},
             refit='recall', scoring={'f1', 'precision', 'accuracy', 'recall'})
```

[24]:
```
# Examine best recall score
### YOUR CODE HERE ###
rf_cv.best_score_
```

[24]:
```
0.9908534395531852
```

[25]:
```
def make_results(model_name, model_object):
    # Get all the results from the CV and put them in a df
    cv_results = pd.DataFrame(model_object.cv_results_)

    # Isolate the row of the df with the max(mean f1 score)
    best_estimator_results = cv_results.iloc[cv_results['mean_test_f1'].
 ↪idxmax(), :]

    # Extract accuracy, precision, recall, and f1 score from that row
    f1 = best_estimator_results.mean_test_f1
    recall = best_estimator_results.mean_test_recall
    precision = best_estimator_results.mean_test_precision
    accuracy = best_estimator_results.mean_test_accuracy

    # Create table of results
    table = pd.DataFrame({'Model': [model_name],
                          'F1': [f1],
                          'Recall': [recall],
                          'Precision': [precision],
                          'Accuracy': [accuracy]
                         }
                        )

    return table
```

[26]:
```
# Get all the results from the CV and put them in a df
# Isolate the row of the df with the max(mean precision score)
### YOUR CODE HERE ###

rf_cv_results = make_results('Random Forest Tuned', rf_cv)
print(rf_cv_results)
```

```
                 Model        F1    Recall  Precision  Accuracy
0  Random Forest Tuned  0.995143  0.990853   0.999479  0.995109
```

[27]:
```
# Examine best parameters
### YOUR CODE HERE ###
rf_cv.best_params_
```

[27]:
```
{'max_depth': 5,
 'max_features': 0.6,
 'max_samples': 0.7,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 75}
```

Question: How well is your model performing? Consider average recall score and precision score.

Answer: With an average recall score of 0.991 over five cross-validation folds, this model performs exceptionally well. Examining the precision score shows that it does not classify all samples as claims, making its classifications nearly perfect.

4.3.6 Build an XGBoost model

[28]:
```
# Instantiate the XGBoost classifier
### YOUR CODE HERE ###
xgb = XGBClassifier(objective='binary:logistic', random_state=0)

# Create a dictionary of hyperparameters to tune
### YOUR CODE HERE ###
cv_params = {'max_depth': [4,8,12],
             'min_child_weight': [3, 5],
             'learning_rate': [0.01, 0.1],
             'n_estimators': [300, 500]
             }

# Define a dictionary of scoring metrics to capture
### YOUR CODE HERE ###
scoring = {'accuracy', 'precision', 'recall', 'f1'}

# Instantiate the GridSearchCV object
### YOUR CODE HERE ###
```

```
xgb_cv = GridSearchCV(xgb, cv_params, scoring=scoring, cv=5, refit='recall',␣
 ↪n_jobs=-1)
```

[29]:
```
%%time
# Fit model with training set
xgb_cv.fit(X_train, y_train)
```

```
CPU times: user 3min 11s, sys: 439 ms, total: 3min 11s
Wall time: 4min 11s
```

[29]:
```
GridSearchCV(cv=5,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     callbacks=None, colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None,
                                     early_stopping_rounds=None,
                                     enable_categorical=False, eval_metric=None,
                                     feature_types=None, gamma=None,
                                     gpu_id=None, grow_policy=None,
                                     importance_type=None,
                                     interaction_constraints=None,
                                     learning_rate=None,…
                                     max_delta_step=None, max_depth=None,
                                     max_leaves=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     n_estimators=100, n_jobs=None,
                                     num_parallel_tree=None, predictor=None,
                                     random_state=0, …),
             n_jobs=-1,
             param_grid={'learning_rate': [0.01, 0.1], 'max_depth': [4, 8, 12],
                         'min_child_weight': [3, 5],
                         'n_estimators': [300, 500]},
             refit='recall', scoring={'f1', 'precision', 'accuracy', 'recall'})
```

[30]:
```
# Examine best recall score
### YOUR CODE HERE ###
xgb_cv.best_score_
```

[30]:
```
0.9906808769992594
```

[31]:
```
# Get all the results from the CV and put them in a df
# Isolate the row of the df with the max(mean precision score)
### YOUR CODE HERE ###

xgb_cv_results = make_results('XGBoost Tuned', xgb_cv)
print(xgb_cv_results)
```

```
           Model        F1    Recall  Precision  Accuracy
```

```
0  XGBoost Tuned  0.995142  0.990681   0.999652  0.995109
```

[32]:
```
# Examine best parameters
### YOUR CODE HERE ###
xgb_cv.best_params_
```

[32]:
```
{'learning_rate': 0.1,
 'max_depth': 4,
 'min_child_weight': 5,
 'n_estimators': 300}
```

Question: How well does your model perform? Consider recall score and precision score.

Answer: This model demonstrates excellent performance. While its recall score is marginally lower than that of the random forest model, it achieves a perfect precision score.

4.4 PACE: Execute

Consider the questions in your PACE Strategy Document to reflect on the Execute stage.

4.4.1 Task 7. Evaluate model

Evaluate models against validation criteria.

Random forest

[33]:
```
# Use the random forest "best estimator" model to get predictions on the␣
 ↪encoded testing set
### YOUR CODE HERE ###
y_pred = rf_cv.best_estimator_.predict(X_val)
```

Display the predictions on the encoded testing set.

[34]:
```
# Display the predictions on the encoded testing set
### YOUR CODE HERE ###
print(y_pred)
```

```
[1 0 1 … 1 1 1]
```

Display the true labels of the testing set.

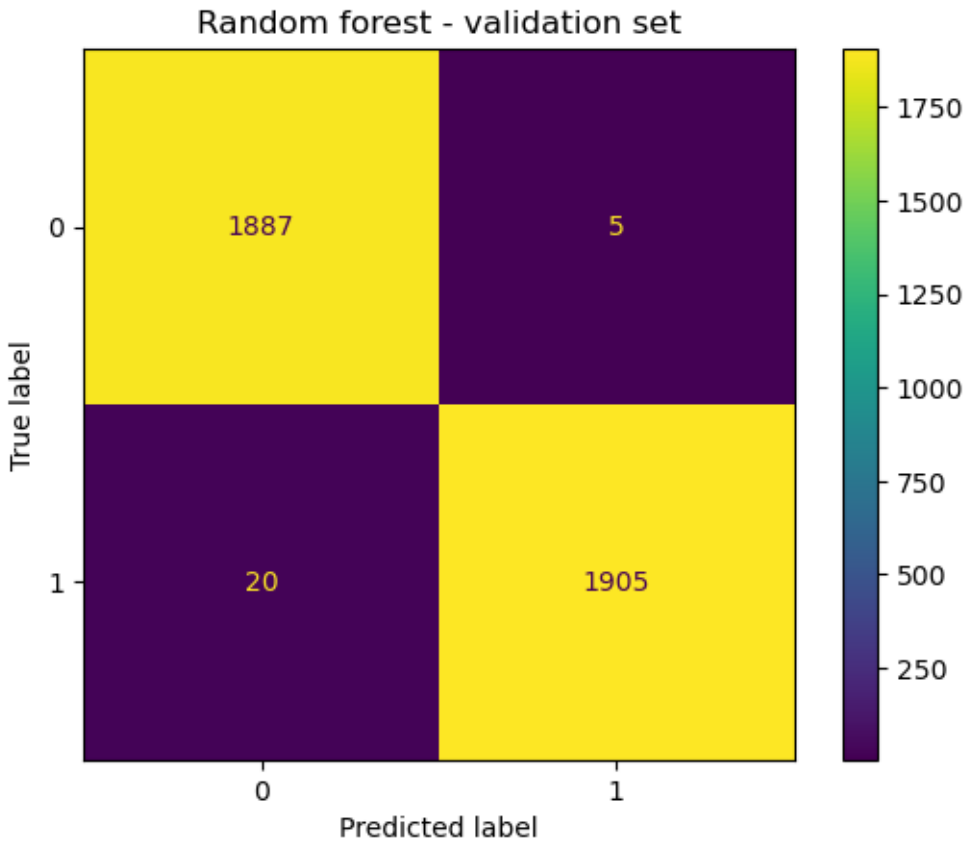
[35]:
```
# Display the true labels of the testing set
### YOUR CODE HERE ###
print(y_val)
```

```
5846     1
12058    0
2975     1
8432     1
6863     1
        ..
```

```
6036     1
6544     1
2781     1
6426     1
4450     1
Name: claim_status, Length: 3817, dtype: int64
```

Create a confusion matrix to visualize the results of the classification model.

[36]:
```
# Create a confusion matrix to visualize the results of the classification model

# Compute values for confusion matrix
### YOUR CODE HERE ###
log_cm = confusion_matrix(y_val, y_pred)

# Create display of confusion matrix
### YOUR CODE HERE ###
log_disp = ConfusionMatrixDisplay(confusion_matrix = log_cm, display_labels =␣
 ↪None)

# Plot confusion matrix
### YOUR CODE HERE ###
log_disp.plot()

# Display plot
### YOUR CODE HERE ###
plt.title('Random forest - validation set');
plt.show()
```



Create a classification report that includes precision, recall, f1-score, and accuracy metrics to evaluate the performance of the model.

[37]:
```
# Create a classification report
# Create classification report for random forest model
### YOUR CODE HERE ###
target_labels = ['opinion', 'claim']
print(classification_report(y_val, y_pred, target_names=target_labels))
```

```
              precision    recall  f1-score   support

     opinion       0.99      1.00      0.99      1892
       claim       1.00      0.99      0.99      1925

    accuracy                           0.99      3817
   macro avg       0.99      0.99      0.99      3817
weighted avg       0.99      0.99      0.99      3817
```

Question: What does your classification report show? What does the confusion matrix indicate?

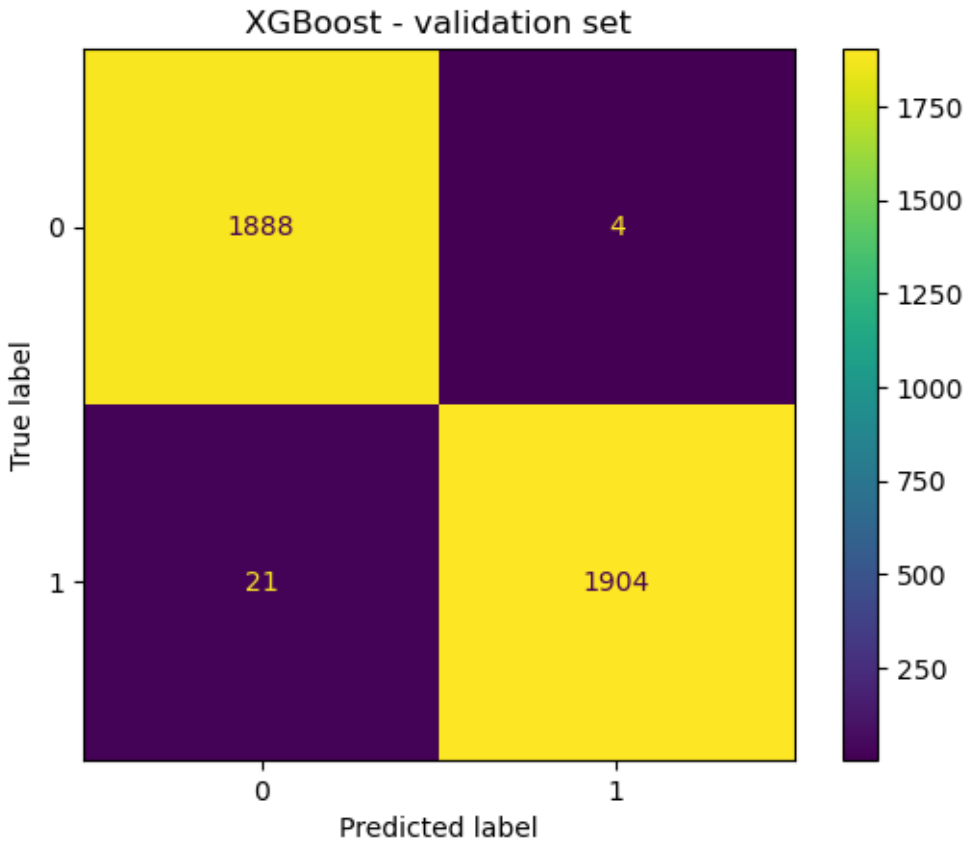
Answer: According to the classification report, the random forest model's scores were almost

flawless. The confusion matrix reveals 10 misclassifications, with five false positives and five false negatives.

XGBoost

[38]:
```
#Evaluate XGBoost model
### YOUR CODE HERE ###
y_pred = xgb_cv.best_estimator_.predict(X_val)
```

[39]:
```
# Compute values for confusion matrix
### YOUR CODE HERE ###
log_cm = confusion_matrix(y_val, y_pred)

# Create display of confusion matrix
### YOUR CODE HERE ###
log_disp = ConfusionMatrixDisplay(confusion_matrix=log_cm, display_labels=None)

# Plot confusion matrix
### YOUR CODE HERE ###
log_disp.plot()

# Display plot
### YOUR CODE HERE ###
plt.title('XGBoost - validation set');
plt.show()
```



[40]:
```
# Create a classification report
### YOUR CODE HERE ###
target_labels = ['opinion', 'claim']
print(classification_report(y_val, y_pred, target_names=target_labels))
```

```
              precision    recall  f1-score   support

     opinion       0.99      1.00      0.99      1892
       claim       1.00      0.99      0.99      1925

    accuracy                           0.99      3817
   macro avg       0.99      0.99      0.99      3817
weighted avg       0.99      0.99      0.99      3817
```

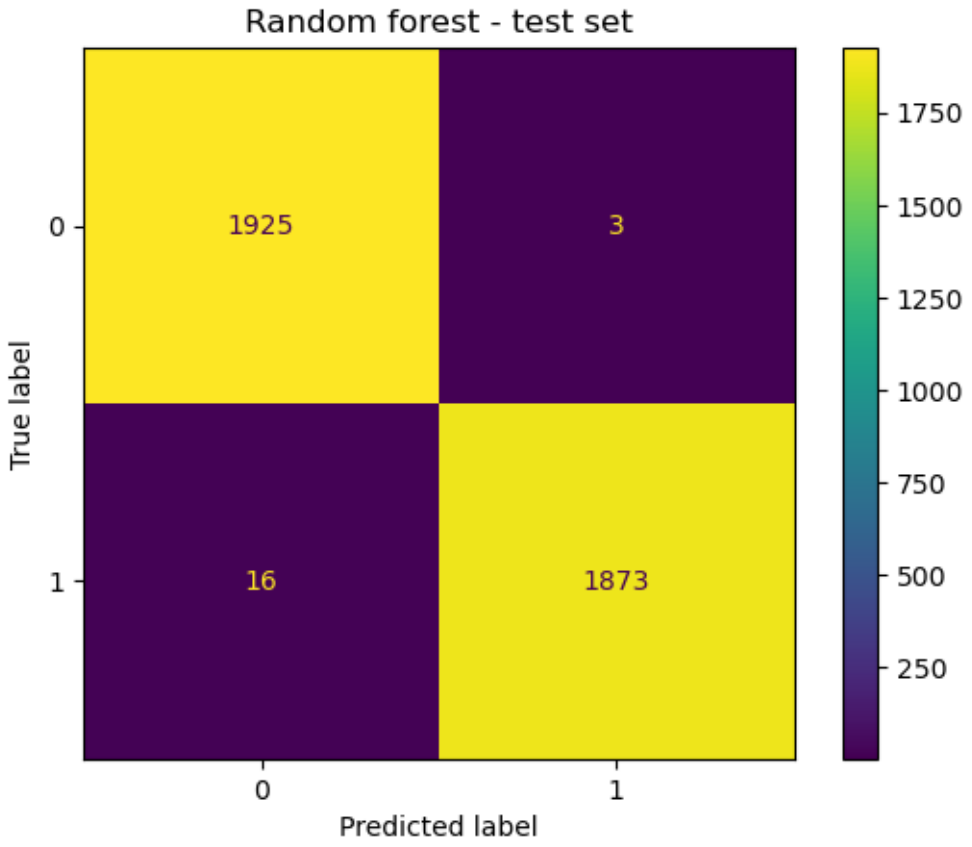
Question: Describe your XGBoost model results. How does your XGBoost model compare to your random forest model?

Answer: The XGBoost model's results were nearly perfect too, but it made more false negative errors. Since it's important to catch all claim videos, the random forest model with a better recall score is the winner.

4.4.2 Use champion model to predict on test data

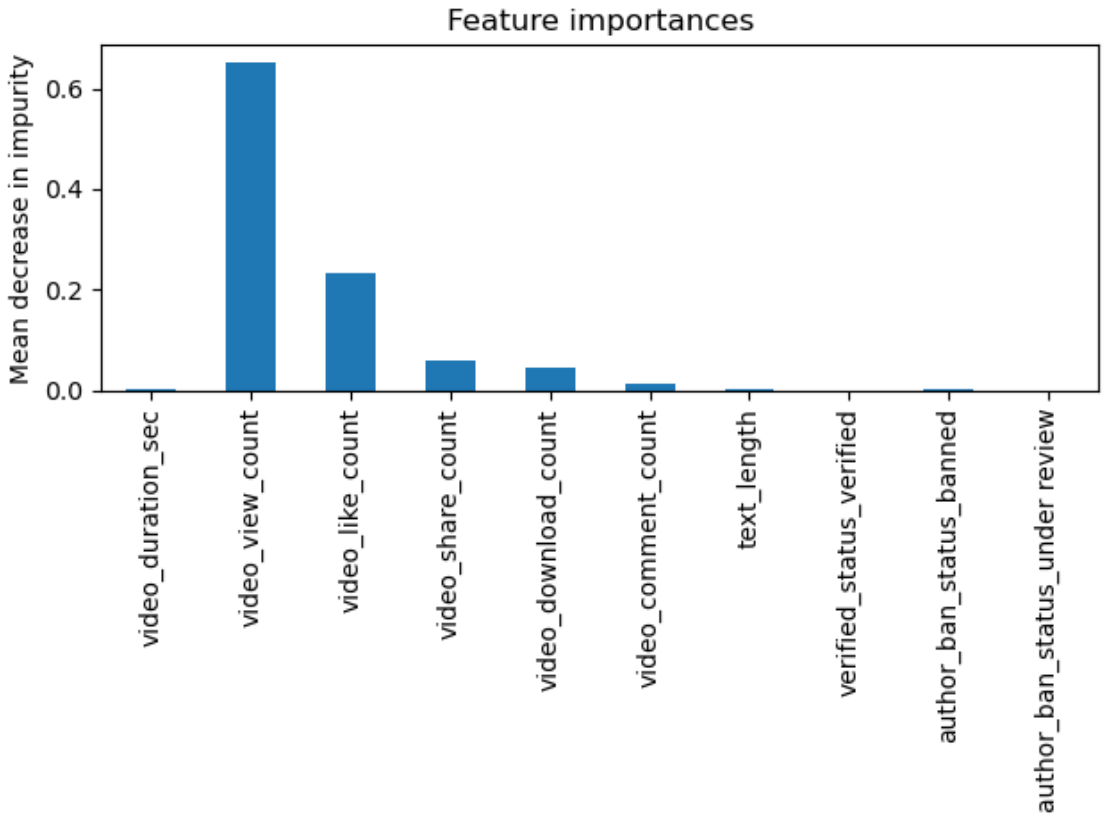
[41]:
```
### YOUR CODE HERE ###
y_pred = rf_cv.best_estimator_.predict(X_test)
```

[42]:
```
# Compute values for confusion matrix
log_cm = confusion_matrix(y_test, y_pred)

# Create display of confusion matrix
log_disp = ConfusionMatrixDisplay(confusion_matrix=log_cm, display_labels=None)

# Plot confusion matrix
log_disp.plot()

# Display plot
plt.title('Random forest - test set');
plt.show()
```



Feature importances of champion model

[43]:
```
### YOUR CODE HERE ###
importances = rf_cv.best_estimator_.feature_importances_
rf_importances = pd.Series(importances, index=X_test.columns)

fig, ax = plt.subplots()
rf_importances.plot.bar(ax=ax)
ax.set_title('Feature importances')
ax.set_ylabel('Mean decrease in impurity')
fig.tight_layout()
```



Question: Describe your most predictive features. Were your results surprising?

Answer: All the top predictive features were associated with the video's engagement levels. This is not surprising, given the conclusions drawn from previous EDA.

4.4.3 Task 8. Conclusion

In this step use the results of the models above to formulate a conclusion. Consider the following questions:

1. **Would you recommend using this model? Why or why not?**
2. **What was your model doing? Can you explain how it was making predictions?**

3. **Are there new features that you can engineer that might improve model performance?**

4. **What features would you want to have that would likely improve the performance of your model?**

Remember, sometimes your data simply will not be predictive of your chosen target. This is common. Machine learning is a powerful tool, but it is not magic. If your data does not contain predictive signal, even the most complex algorithm will not be able to deliver consistent and accurate predictions. Do not be afraid to draw this conclusion.

Answers:

1. Yes, this model is recommended because it excelled in both validation and test holdout data. Additionally, it consistently achieved high precision and F1 scores, effectively classifying claims and opinions.
2. The model's primary task was to classify videos using the most predictive features, which were tied to user engagement metrics such as views, likes, shares, and downloads.
3. The model's current performance is nearly flawless, so additional feature engineering is unnecessary.
4. The model doesn't need additional features; however, data on the number of times each video was reported and the total number of user reports for all an author's videos would be helpful.

Congratulations! You've completed this lab. However, you may not notice a green check mark next to this item on Coursera's platform. Please continue your progress regardless of the check mark. Just click on the "save" icon at the top of this notebook to ensure your work has been logged.