To Skip or Not to Skip?

Final report

Maira Januzzi - Beatrice Tomasello

Executive Summary.

Spotify is a media services provider, founded in 2006 by Daniel Elk and Martin Lorentzon. It provides DRM-protected music, videos and podcasts from record labels and media companies. Spotify gives you access to millions of songs, podcasts and videos from artists all over the world.Spotify is simple because you can just access content for free by signing up using an email address.

Being Spotify a “freemium” service, you can find differences between Spotify Free and Premium: Free version is ad-supported, like radio stations, it can be accessed from computer and mobile phone, but if you want a full service you need to get and pay for a Premium subscription. One of the most relevant differences is that in the free version you have limited features and you are allowed to skip up to six times per hour, every hour. on the other hand Premium users have access to everything, they can play any song (on demand), as well as find and hear playlists, create playlists, listen offline, hear high - quality music and, above all, they can skip as many songs as they like, without ads.

The goal for our models is to predict the behavior user for skipping or not a song, using as our Y the variable ‘Not_skipped’, understanding if and how much our output variable is dependent on the features included in our dataset. While there is a large related body of work on recommender systems, there is very little work, or data, describing how users sequentially interact with the streamed content they are presented with. In particular within music, the question of if, and when, a user skips a track is an important implicit feedback signal. In order to get this feedback we developed our analysis and prediction based on supervised machine learning models: we used Logistic Regression, which we concluded applying Ridge and Lasso regression, and the Regression Tree model, together with the Random Forest.

Dataset.

We found the dataset for our project on AIC (Artificial Intelligence crowd), where it has been presented for challenge in 2019:
https://www.aicrowd.com/challenges/spotify-sequential-skip-prediction-challenge.
Our initial dataset is composed of 99999 observations for 21 variables, of which these are the provided explanations:

session_id	unique identifier for the session that this row is a part of
session_position	position of row within session

session_length	number of rows in session
track_id_clean	unique identifier for the track played. This is linked with track_id in the track features and metadata table.
skip_1	Boolean indicating if the track was only played very briefly
skip_2	Boolean indicating if the track was played briefly
skip_3	Boolean indicating if the track was almost fully played
not_skipped	Boolean indicating if the track was played in its entirety
context_switch	Boolean indicating if the user changed context between the previous row and the current row. This could for example occur if the user switched from one playlist to another.
no_pause_before_play	Boolean indicating if there was no pause between playback of the previous track and this track
short_pause_before_play	Boolean indicating if there was a short pause between playback of the previous track and this track
long_pause_before_play	Boolean indicating if there was a long pause between playback of the previous track and this track
hist_user_behavior_n_seekfwd	Number of times the user did a seek forward within track
hist_user_behavior_n_seekback	Number of times the user did a seek back within track
hist_user_behavior_is_shuffle	Boolean indicating if the user encountered this track while shuffle mode was activated
hour_of_day	The hour of day
date	The date
premium	Boolean indicating if the user was on premium or not.
context_type	what type of context the playback occurred within
hist_user_behavior_reason_start	the user action which led to the current track being played
hist_user_behavior_reason_end	the user action which led to the current track playback ending.

In this dataset we do not have any missing values. We decided to drop ‘hour_of_day’ and ‘date’ because during data visualization we discovered that they didn’t influence our output. We also got dummies for every variable which presented different classes. As asked from the challenge organizers we cite the following paper: @inproceedings{brost2019music, title={The Music Streaming Sessions Dataset}, author={Brost, Brian and Mehrotra, Rishabh and Jehan, Tristan}, booktitle={Proceedings of the 2019 Web Conference}, year={2019}, organization={ACM} }

Analysis.

1 - Logistic Regression.

Since our variables are represented in the cleaned dataset as binary, the first model we decided to apply is Logistic Regression. We used the built-in feature importance method and then the Variance Inflation Factors to understand which variables were to be kept or discarded. We performed Ridge Regression and Lasso as well, and we found out that our accuracy underwent non relevant changes switching from no penalty, to l1 and l2.
As our results for this model we got: Accuracy on train and test for Logistic: 0.99
Ridge and Lasso: 0.98.

2 - Regression Tree.

A decision tree is a managerial tool that presents all the decision alternatives and outcomes in a flowchart type of diagram. Each branch of the tree represents a decision option, its cost and the probability that it is likely to occur. A decision tree illustrates graphically all the possible alternatives, probabilities and outcomes and identifies the benefits of using decision analysis.
As our results for this model we got:
Root Mean Squared Error on train set: 0.09882687922411729
Root Mean Squared Error on test set: 0.10529649555232867
Mean of y_train: 0.3387333873338733

3 - Random Forest.

In the end we applied the Random Forest consists in a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees. Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.
This are the results we got for this model:
Root Mean Squared Error on train set: 0.08721280950953403
Root Mean Squared Error on test set: 0.1051392623351552
Mean of y_train: 0.3387333873338733
Accuracy of RF classifier on training set: 0.99
Accuracy of RF classifier on test set: 0.99

Conclusions.

First when we were exploring the data we saw that for our dependent variable had as percentage of song skipped about 66% and fully played 34%. Comparing all the models that we use we got the same or similar accuracy = 0,99, for all of them. With the decision tree we can visualize better our results, and understand which variables affect on the result.

Technical Appendix.

Codes

Data visualization.

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt
plt.rc("font", size=14)
import seaborn as sns
sns.set(style="white")
sns.set(style="whitegrid", color_codes=True)
#import the dataset, check size and columns
data = pd.read_excel (r'C:\Users\panda\python\Spotify_Dataset.xlsx', index_col=1)
data = data.dropna()
print(data.shape)
print(list(data.columns))
print(data.ndim)
data.head()
data.dtypes
data['not_skipped'].value_counts()
sns.countplot(x='not_skipped', data=data, palette='hls')
plt.show()
plt.savefig('count_plot')
count_skipped=len(data[data['not_skipped']==False])
count_played=len(data[data['not_skipped']==True])
pct_skipped=count_skipped/(count_skipped+count_played)
print('the percentage of songs skipped is', pct_skipped*100)
pct_played=count_played/(count_played+count_skipped)
print('the percentage of songs fully played is', pct_played*100)
data.groupby('not_skipped').mean()
%matplotlib inline
pd.crosstab(data.context_type,data.not_skipped).plot(kind='bar')
plt.title('Skipping Behavior depending on Context')
plt.xlabel('Context')
plt.ylabel('Skipping Behavior')
plt.savefig('skip_per_context')
#we can see it depends quite a bit on the context, with high skipping especially within the
user library
pd.crosstab(data.premium, data.not_skipped).plot(kind='bar')
plt.title('Skipping Behavior for Premium and non Premium users')
plt.xlabel('Premium')
plt.ylabel('Skippin Behavior')
plt.savefig('ski_premium')
#we can observe a strongly different outcome based on the subscription
data['premium'].value_counts()
sns.countplot(x='premium', data=data, palette='hls')
plt.show()
```

plt.savefig('premium_users')
#to explain the difference between skippng behavior we take into account the distribution of premium users
pd.crosstab(data.no_pause_before_play, data.not_skipped).plot(kind='bar')
plt.title('Skipping Behavior within session')
plt.xlabel('Pause before play')
plt.ylabel('Skippin Behavior')
plt.savefig('ski_premium')
#we can observe that after a pause the songs are skipped more
sns.countplot(x='no_pause_before_play', data=data, palette='hls')
plt.show()
plt.savefig('no_pause_before_play')
#much more pauses
pd.crosstab(data.long_pause_before_play, data.not_skipped).plot(kind='bar')
plt.title('Skipping Behavior after long pause')
plt.xlabel('Long pause before play')
plt.ylabel('Skippin Behavior')
plt.savefig('skip_long_pause')
#less likely to skip after
pd.crosstab(data.hour_of_day, data.not_skipped).plot(kind='bar')
plt.title('Skipping Behavior during the day')
plt.xlabel('Hour of the Day')
plt.ylabel('Skipping Behavior')
plt.savefig('skip_hour_day')
#we can observe a clearly increasing trend from 7 that reaches the peak at 17 and then decreases again, let's how the usage is distributed
sns.countplot(x='hour_of_day', data=data, palette='hls')
plt.show()
plt.savefig('hours')
#we observe that the usage during the day totally reflects the skipping behavior, thus hour of the day is not a good predictor
pd.crosstab(data.date, data.not_skipped).plot(kind='bar')
plt.title('Skipping Behavior for date')
plt.xlabel('Date')
plt.ylabel('Skipping Behavior')
plt.savefig('skip_date')
sns.countplot(x='date', data=data, palette='hls')
plt.show()
plt.savefig('date')
#same here, date is nor relevant again

Logistic Regression.

#import the libraries
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import preprocessing

```
from sklearn.linear_model import LogisticRegression
#import dataset
#print dataset details
data=pd.read_excel('Spotify Dataset.xlsx', index_col=1)
data=data.dropna()
print(data.shape)
print(list(data.columns))
print(data.ndim)
#print first taste of the dataset
data.head()
#show the data types of the variables
data.dtypes
#check if there are null values
print('we have ' ,len(data), ' rows')
data.isnull().sum().sort_values(ascending = False)
#get rid of the variables that we don't need. using inplace we modify the original dataset
data.drop(['session_id', 'session_length', 'track_id_clean', 'skip_1', 'skip_2', 'skip_3',
'hour_of_day', 'date'], axis=1, inplace=True)
#show the modified dataset
data.head()
#turn the booleans into int variables
data[['not_skipped', 'hist_user_behavior_is_shuffle', 'premium']]=(data[['not_skipped',
'hist_user_behavior_is_shuffle', 'premium']]==True).astype(int)
#get the dummies for the last three columns of strings
dummy = pd.get_dummies(data[['context_type',
'hist_user_behavior_reason_start',
'hist_user_behavior_reason_end']], drop_first = True)
dummy.head()
#build the final dataset with the dummies
#dropping the original variables that we turned into dummies
df = pd.concat([data, dummy], axis=1)
df.drop(['context_type',
'hist_user_behavior_reason_start',
'hist_user_behavior_reason_end'], inplace=True, axis=1)
df.head()
#correlation
corrmat = df.corr()
corrmat
#show the heatmap for correlation
plt.figure(figsize=(20,20))
sns.heatmap(corrmat, cmap ="YlGnBu", linewidths = 0.1)
plt.show()
#check for multicollinearity
corr = df.drop('not_skipped', axis=1).corr()
plt.figure(figsize=(12, 10))
sns.heatmap(corr[(corr >= 0.5) | (corr <= -0.4)],
cmap='viridis', vmax=1.0, vmin=-1.0, linewidths=0.1,
annot=True, annot_kws={"size": 8}, square=True);
df.drop('long_pause_before_play', axis=1, inplace= True)
#explore  not_skipped which is gonna be our target variable
df['not_skipped'].describe()
```

```
#let's have a look at how our variable is composed
#our y is pretty much balanced, we don't need undersampling
df['not_skipped'].value_counts()
sns.countplot(x='not_skipped', data=df, palette= 'hls')
plt.show()
plt.savefig('Skipping_Behaviour')
#continue the data exploring getting some percentages
count_skipped=len(data[df['not_skipped']==False])
count_played=len(data[df['not_skipped']==True])
pct_skipped=count_skipped/(count_skipped+count_played)
print('the percentage of songs skipped is', pct_skipped*100)
pct_played=count_played/(count_played+count_skipped)
print('the percentage of songs fully played is', pct_played*100)
data = df
X = df.iloc[:,1:]  #independent columns
y = df['not_skipped']    #target column i.e price range
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt
model = ExtraTreesClassifier()
model.fit(X,y)
print(model.feature_importances_) #use inbuilt class feature_importances of tree based
classifiers
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
data_fin = df[['not_skipped','hist_user_behavior_reason_start_backbtn',
'hist_user_behavior_reason_start_clickrow', 'no_pause_before_play',
'hist_user_behavior_n_seekback','short_pause_before_play','hist_user_behavior_n_seekfwd
',
'hist_user_behavior_reason_end_endplay','hist_user_behavior_reason_start_trackdone','hist
_user_behavior_reason_end_fwdbtn','hist_user_behavior_reason_end_trackdone']]
data_fin.head()
#then VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant
VIF_set = data_fin.copy().drop(columns=['not_skipped'])
cols=VIF_set.columns
VIF_set = add_constant(VIF_set.values)
pd.Series(["{0:.2f}".format(variance_inflation_factor(VIF_set, i)) for i in
range(VIF_set.shape[1])], index=['constant'] + list(cols))
#define my target variable and the regressors
X = data_fin.drop('not_skipped', axis=1)
y = data_fin['not_skipped']
print(X.shape)
print(y.shape)
#split in train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
```

```
print(y_test.shape)
# check target distribution of train and test set
plt.figure(figsize=(10,6))
sns.distplot(y_train, label='train')
sns.distplot(y_test, label='test')
plt.legend(fontsize=15)
plt.show()
# fit the model
import statsmodels.api as sm
logit_model=sm.Logit(y_train,X_train)
result=logit_model.fit()
print(result.summary2())
# fit the model on training set
model = LogisticRegression(solver='lbfgs', random_state=0, penalty='none', max_iter=1)
model.fit(X_train, y_train) # training the algorithm
model = LogisticRegression(solver='lbfgs', random_state=0, penalty='none')
print(model.fit(X_train, y_train)) # training the algorithm
print('\nNumber of iterations: ' + str(model.n_iter_[0]) + '/' + str(model.max_iter))
# get coefficients
print('Intercept:', model.intercept_)
print('Slope:', model.coef_)
pd.DataFrame({'Variable': ['intercept'] + list(X.columns),
'Coefficient': ["{0:.5f}".format(v) for v in
np.append(model.intercept_,model.coef_.flatten()).round(6)]})
X_train, y_train = np.array(y_train), np.array(X_train)
y_train.reshape(1, -1)
# get fitted value on training set
y_train_predicted = model.predict(X_train)
# compare predictions
display(pd.DataFrame({'True': y_train.flatten(), 'Predicted': y_train_predicted.flatten()}))
# compare predicted probabilities (default threshold for converting to 0 or 1 is 0.5)
y_train_predicted_prob = model.predict_proba(X_train)[:,1]
display(pd.DataFrame({'True': y_train.flatten(), 'Predicted_prob':
y_train_predicted_prob.flatten(), 'Predicted': y_train_predicted.flatten()}))
# evaluate confusion matrix
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
def plot_confusion_matrix(y_true, y_pred,
normalize=False,
title=None,
cmap=plt.cm.Blues):
"""
This function prints and plots the confusion matrix.
Normalization can be applied by setting `normalize=True`.
"""
if not title:
if normalize:
title = 'Normalized confusion matrix'
else:
title = 'Confusion matrix, without normalization'
# Compute confusion matrix
```

```
cm = confusion_matrix(y_true, y_pred)
# Only use the labels that appear in the data
classes = ['0', '1']
if normalize:
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
print("Normalized confusion matrix")
else:
print('Confusion matrix, without normalization')
print(cm)
fig, ax = plt.subplots()
im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
ax.figure.colorbar(im, ax=ax)
# We want to show all ticks...
ax.set(xticks=np.arange(cm.shape[1]),
yticks=np.arange(cm.shape[0]),
# ... and label them with the respective list entries
xticklabels=classes, yticklabels=classes,
title=title,
ylabel='True label',
xlabel='Predicted label')
# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
rotation_mode="anchor")
# Loop over data dimensions and create text annotations.
fmt = '.2f' if normalize else 'd'
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
for j in range(cm.shape[1]):
ax.text(j, i, format(cm[i, j], fmt),
ha="center", va="center",
color="white" if cm[i, j] > thresh else "black")
fig.tight_layout()
return ax
np.set_printoptions(precision=2)
# Plot non-normalized confusion matrix
plot_confusion_matrix(y_train, y_train_predicted)
plt.show()
# evaluate precision, recall, F1-score on train set
from sklearn.metrics import classification_report
print(classification_report(y_train, y_train_predicted))
# evaluate ROC curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_train, y_train_predicted)
fpr, tpr, thresholds = roc_curve(y_train, y_train_predicted_prob)
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
from sklearn.metrics import precision_recall_curve
precision, recall, thresholds = precision_recall_curve(y_train, y_train_predicted_prob)
pr_auc = metrics.auc(recall, precision)
from sklearn.metrics import precision_recall_curve
precision, recall, thresholds = precision_recall_curve(y_train, y_train_predicted_prob)
pr_auc = metrics.auc(recall, precision)
plt.title("Precision-Recall vs Threshold Chart")
plt.plot(thresholds, precision[: -1], "b--", label="Precision")
plt.plot(thresholds, recall[: -1], "r--", label="Recall")
plt.ylabel("Precision, Recall")
plt.xlabel("Threshold")
plt.legend(loc="lower left")
plt.ylim([0,1])
# evaluate performance on test set
y_test_predicted = model.predict(X_test)
y_test_predicted_prob = model.predict_proba(X_test)[:,1]
plot_confusion_matrix(y_test, y_test_predicted)
plt.show()
print(classification_report(y_test, y_test_predicted))
logit_roc_auc = roc_auc_score(y_test, y_test_predicted)
fpr, tpr, thresholds = roc_curve(y_test, y_test_predicted_prob)
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
precision, recall, thresholds = precision_recall_curve(y_test, y_test_predicted_prob)
pr_auc = metrics.auc(recall, precision)
plt.title("Precision-Recall vs Threshold Chart")
plt.plot(thresholds, precision[: -1], "b--", label="Precision")
plt.plot(thresholds, recall[: -1], "r--", label="Recall")
plt.ylabel("Precision, Recall")
plt.xlabel("Threshold")
plt.legend(loc="lower left")
plt.ylim([0,1])
from sklearn.metrics import precision_recall_curve
precision, recall, thresholds = precision_recall_curve(y_test, y_test_predicted_prob)
pr_auc = metrics.auc(recall, precision)

plt.title("Precision-Recall vs Threshold Chart")
plt.plot(thresholds, precision[: -1], "b--", label="Precision")
plt.plot(thresholds, recall[: -1], "r--", label="Recall")
```

```
plt.from sklearn.linear_model import LogisticRegressionCV
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
def fit_model(X_input_train, X_input_test, y_train, y_test, penalty, n_fold):
if penalty=='none':
model = LogisticRegression(solver='lbfgs', random_state=0, penalty=penalty)
model.fit(X_input_train, y_train) # training the algorithm
else:
model = LogisticRegressionCV(solver='liblinear', cv=n_fold, random_state=0,
penalty=penalty)
model.fit(X_input_train, y_train) # training the algorithm
lambda_set=model.Cs_
best_lambda=model.C_
best_lambda_index=np.where(lambda_set == best_lambda)[0]
coeff_paths=model.coefs_paths_[1]
# plot coeff paths
coeff_paths=coeff_paths.mean(axis=0)
avg_accuracy_cv=model.scores_[1].mean(axis=0)
y_axis_range=[coeff_paths.min()*1.2, coeff_paths.max()*1.2]
fig, ax1 = plt.subplots(figsize=(15,15))
for i in range(0,coeff_paths.shape[1]):
if i<coeff_paths.shape[1]-1:
var_lab=X_input_train.columns[i]
else:
var_lab='intercept'
ax1.plot(range(0,len(lambda_set)), coeff_paths[:,i], label=var_lab)
ax1.tick_params(axis='y', labelcolor='black', labelsize=20)
ax1.set_ylabel('coefficients', color='black', fontsize=25)
ax1.set_xlabel('lambda', color='black', fontsize=25)
ax1.set_xticks(range(0,len(lambda_set)))
ax1.set_xticklabels(lambda_set.round(5), color='black', fontsize=15, rotation=45)
ax1.legend(loc='center left', bbox_to_anchor=(1.2, 0.5), fontsize=22, ncol=2)
ax1.set_title('\nCoefficients magnitude vs lambda values\n', fontsize=35)
ax2 = ax1.twinx()
ax2.set_ylabel('Cross-Validated Accuracy', color='dodgerblue', fontsize=25)
ax2.plot((avg_accuracy_cv*100).round(4), color='dodgerblue', linestyle='--', linewidth=7,
ax2.tick_params(axis='y', labelcolor='dodgerblue', labelsize=20)
ax2.set_yticklabels(['{:,.2%}'.format(x) for x in avg_accuracy_cv])
# vertical line corresponding to best Lambda
ax2.axvline(best_lambda_index, color='red', linestyle='--', linewidth=7, label='best Lambda')
ax2.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15), fontsize=25)
plt.show()
coeff=pd.DataFrame({'Variable': ['intercept'] + list(X_input_train.columns),
'Coefficient': ["{0:.8f}".format(v) for v in
np.append(model.intercept_,model.coef_.flatten()).round(6)]})
display(coeff)
y_train_predicted = model.predict(X_input_train)
y_train_predicted_prob = model.predict_proba(X_input_train)[:,1]
y_test_predicted = model.predict(X_input_test)
y_test_predicted_prob = model.predict_proba(X_input_test)[:,1]
```

```
train_accuracy=accuracy_score(y_train, y_train_predicted)
test_accuracy=accuracy_score(y_test, y_test_predicted)
train_precision=precision_score(y_train, y_train_predicted, average='macro')
test_precision=precision_score(y_test, y_test_predicted, average='macro')
train_recall=recall_score(y_train, y_train_predicted, average='macro')
test_recall=recall_score(y_test, y_test_predicted, average='macro')
results = pd.DataFrame({'Penalty': [penalty],
'Train_Accuracy': [train_accuracy], 'Test_Accuracy': [test_accuracy],
'Train_Precision': [train_precision], 'Test_Precision': [test_precision],
'Train_Recall': [train_recall], 'Test_Recall': [test_recall]})
display(results)
return resultsylabel("Precision, Recall")
plt.xlabel("Threshold")
plt.legend(loc="lower left")
plt.ylim([0,1])
X_orig = df.drop(columns=['not_skipped'])
y_orig = df['not_skipped'].values
print(X.shape)
print(y.shape)
#split in train and test
X_train_orig, X_test_orig, y_train_orig, y_test_orig = train_test_split(X_orig, y_orig,
test_size=0.2, random_state=1)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
# fit LASSO on original dataset with all variables (so that some will be set to 0)
results_LASSO = fit_model(X_train_orig, X_test_orig, y_train, y_test, penalty='l1', n_fold=5)
# fit Ridge on original dataset with all variables (so that some will be shrinked 0)
results_Ridge = fit_model(X_train_orig, X_test_orig, y_train, y_test, penalty='l2', n_fold=5)
# fit LogisticRegression on dataset with selected variables
results_NoPen = fit_model(X_train, X_test, y_train, y_test, penalty='none', n_fold=5)
# compare performance
pd.concat([results_NoPen, results_LASSO, results_Ridge], axis=0)
```

Decision Tree.

```
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn import preprocessing
%matplotlib inline
#import dataset
data=pd.read_excel('Spotify Dataset.xlsx', index_col=1)
data=data.dropna()
#get rid of the variables
```

```
data.drop(['session_id', 'session_length', 'track_id_clean', 'skip_1', 'skip_2', 'skip_3',
'hour_of_day', 'date'], axis=1, inplace=True)
[{"metadata":{"trusted":true},"cell_type":"code","source":"#transform the booleans in 0 and
1\ndata[['not_skipped', 'hist_user_behavior_is_shuffle', 'premium']]=(data[['not_skipped',
'hist_user_behavior_is_shuffle',
'premium']]==True).astype(int)","execution_count":null,"outputs":[]}]
data.head()
#dummies
dummy = pd.get_dummies(data[['context_type',
'hist_user_behavior_reason_start',
'hist_user_behavior_reason_end']], drop_first = True)
dummy.head()
#get rid of the columns that I have turned into dummies
step = pd.concat([data, dummy], axis=1)
step.drop(['context_type',
'hist_user_behavior_reason_start',
'hist_user_behavior_reason_end'], inplace=True, axis=1)
step.head()
# target variable and the regressors
X = step.iloc[:,1:]
y = step['not_skipped']
# K-fold Cross-Validation function
from sklearn.model_selection import KFold
def kFold_CV(X, y, model, n_fold, _display=True):
# generate folds
folds = KFold(n_splits=n_fold, random_state=0, shuffle=True)
# fit model on each k-1 fold and evaluate performances (errors)
results = pd.DataFrame(columns = ['Split', 'Train size', 'Test size', 'Train R^2', 'Train RMSE',
'Test RMSE'], dtype=float).fillna(0)
fig = plt.figure(figsize=(10,1.5*n_fold))
plot_count=1
split_count=1
model_list={}
for train_index, test_index in folds.split(X, y):
 # define train and test (validation) set
X_split_train = X.iloc[train_index, :]
X_split_test = X.iloc[test_index, :]
y_split_train = y.iloc[train_index, :]
y_split_test = y.iloc[test_index, :]
# plot target variable distribution comparison between split_train and split_test set
ax = fig.add_subplot(math.ceil(n_fold / 3), 3, plot_count)
sns.distplot(y_split_train, label='train', ax=ax)
sns.distplot(y_split_test, label='test', ax=ax)
ax.set_title('Target variable distribution\nsplit ' + str(split_count), fontsize=12)
ax.legend(fontsize=8)
# fit model on train set and get performances on train set
model_fit = model.fit(X_split_train, y_split_train.values.ravel())
y_train_predicted = model.predict(X_split_train)
R2_train = metrics.r2_score(y_split_train, y_train_predicted)
RMSE_train = np.sqrt(metrics.mean_squared_error(y_split_train, y_train_predicted))
model_list['split_'+str(split_count)]=model_fit
```

```
# get performance on test set
y_test_predicted = model.predict(X_split_test)
RMSE_test = np.sqrt(metrics.mean_squared_error(y_split_test, y_test_predicted))
# append results
results=results.append(pd.DataFrame([[split_count, X_split_train.shape[0],
X_split_test.shape[0], R2_train,
RMSE_train, RMSE_test]],
columns=results.columns))
split_count += 1
 plot_count += 1
results['Split']=results['Split'].astype(int)
results['Train size']=results['Train size'].astype(int)
results['Test size']=results['Test size'].astype(int)
if _display==True:
plt.tight_layout()
fig.subplots_adjust(top=0.88)
plt.show()
display(results)
else:
plt.close()
return results, model_list
from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor(random_state=0, max_depth=3)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0,
shuffle=True)
model.fit(X_train, y_train)
print(model)
y_train_predicted = model.predict(X_train)
RMSE_train = np.sqrt(metrics.mean_squared_error(y_train, y_train_predicted))
y_test_predicted = model.predict(X_test)
RMSE_test = np.sqrt(metrics.mean_squared_error(y_test, y_test_predicted))
print('\n\nRoot Mean Squared Error on train set:', RMSE_train)
print('Root Mean Squared Error on test set:', RMSE_test)
print('Mean of y_train:', float(y.mean()))
# plot tree
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(model, out_file=dot_data,
filled=True, rounded=True,
special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
# predicted values are the mean value in each terminal node

pd.DataFrame({'True': y_train.values.flatten(), 'Predicted': y_train_predicted.flatten()})
# features importance
feat_importance = pd.DataFrame({'Variable': X.columns, 'Importance':
model.feature_importances_}).sort_values(by=['Importance'], ascending=False)
```

```
display(feat_importance)
sns.barplot(y='Variable', x='Importance', data=feat_importance)
plt.title('Feature Importance', fontsize=15)
plt.show()
```

Random forest.

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators= 50, random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0,
shuffle=True)
model.fit(X_train, y_train.values.flatten())
print(model)
y_train_predicted = model.predict(X_train)
RMSE_train = np.sqrt(metrics.mean_squared_error(y_train, y_train_predicted))
y_test_predicted = model.predict(X_test)
RMSE_test = np.sqrt(metrics.mean_squared_error(y_test, y_test_predicted))
print('\n\nRoot Mean Squared Error on train set:', RMSE_train)
print('Root Mean Squared Error on test set:', RMSE_test)
print('Mean of y_train:', float(y.mean()))
# access all the fitted trees
model.estimators_
# get features importance and plot
feat_importance = pd.DataFrame({'Variable': X.columns, 'Importance':
model.feature_importances_}).sort_values(by=['Importance'], ascending=False)
display(feat_importance)
sns.barplot(y='Variable', x='Importance', data=feat_importance)
plt.title('Feature Importance', fontsize=15)
plt.show()from sklearn.ensemble import RandomForestClassifier
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)
clf = RandomForestClassifier(max_features = 10, random_state = 0).fit(X_train, y_train)
print('Accuracy of RF classifier on training set: {:.2f}'
.format(clf.score(X_train, y_train)))
print('Accuracy of RF classifier on test set: {:.2f}'
.format(clf.score(X_test, y_test)))
# recall all available parameters for Random Forest
print(RandomForestRegressor())
```