Group S274 (4 members): Mini Project Report

Author(s): S274
Statistical Machine Learning, Uppsala Universitet

1 Introduction

With machine learning it is possible to design a system that can estimate the outcome of a certain problem based on statistical patterns of outcome in data sets. This can prove very useful in a broad spectrum of technologies and businesses. In this Mini Project, a machine learning evaluation method is designed to decide one algorithm estimating which songs from a list Andreas Lindholm will like, based on a set of songs he has already rated with either LIKE or DISLIKE. The data that the system processes are 13 high-level features that describe the characteristics of each song. Based on the set of data as well as Andreas ratings of them, the task at hand is to design an algorithm that estimates ratings of the new songs as accurately as possible. This is possible by processing the data and creating an evaluation of multiple machine learning methods.

2 Models Considered

To design the system so that the estimations will be as accurate as possible several methods were considered, optimized and compared. In the following section, a brief description of each considered method is presented.

2.1 Logistic Regression

The logistic regression model is a modification of the linear regression model. The two mentioned models are applied in totally different cases: the former is used for classification problems, where the boundary is strictly defined and the dependent variable is dichotomous. The latter - as the name reveals - is applied for regression problems, where the dependent variable is unbounded. Furthermore, instead of finding the linear relationship between a dependent variable with independent variables (which is the case for linear regression), the aim with logistic regression is to explain the relationship between the dichotomous dependent variable with the independent variables. One similarity both logistic and linear regression models share is that both learn the parameters for each feature to predict the output. However, the logistic regression model solves the loss function cross-entropy loss to estimate the parameters (and for linear regression the parameters are estimated through using the squared error loss). The parameter considered in the logistic regression model is the solver 'liblinear'. This was useful for the data set considered in this project; it is good for handling smaller data sets and, also, simple classification problems, such as binary classification.

2.2 Linear & Quadratic Discriminant Analysis

Linear Discriminant Analysis (LDA) is a linear classifying method that maximizes the separations between classes by finding linear combinations of the features of the classes. At the same time it can reduce high dimensional data to lower-dimensional spaces. The method uses Bayes' rule to determine the probability that an observation x belongs to a class k. The method assumes that the distribution of x is multivariate Gaussian for each class. The x has different mean values but identical covariances. In that way, the LDA-method uses a probabilistic foundation for its classification. Furthermore, it can be interpreted as a dimensionality reduction method that maximizes the between-class variance while minimizing the within-class variance. This means that the observations within one class are grouped together as tightly as possible while the different classes are kept from each other with a 'safe' distance.

Quadratic Discriminant Analysis (QDA) is a variation of LDA-method in that it functions in a similar way, but allows for separation of data in a non-linear fashion. It also differs from the LDA-method as the QDA-method estimates a separate covariance matrix for each class. Compared to LDA, QDA is more flexible but has a higher risk of overfitting. This means that the bias of the method is lower compared to LDA, but has at the same time a larger variance.

2.3 k-Nearest Neighbours

The k-Nearest Neighbours (k-NN) classifier is a non-parametric method, which means that it does not make assumptions about the distribution of the data as for example LDA and QDA does. Instead it allows the flexibility of the model to form from the available data. The kNN-method estimates the class of an input x based on the classifications of the training data set that is nearest to x. The classification is determined by the majority vote of the k nearest neighbours, which means that depending on how many 'neighbours' the model will take into account the accuracy may vary. Choosing the number of neighbours will affect how if the model over- or underfit the data. Choosing one neighbour ($k = 1$) will certainly overfit and increase the noise in the data, because the model fits the data point which is closest to the 1-nearest neighbour.

2.4 Bootstrap Aggregation

Bagging, or bootstrap aggregation, builds on the idea to reduce the variance for a certain model at the same time as keeping a small bias and is based on the concept of averaging. A central part of bagging, as the name suggests, is bootstrapping. This is a method for creating new data sets from the original data set, with the same size as the original one. It is of importance that the original data set explains the situation in question in a detailed way, since bootstrapping means creating new data sets based on random data from the original set instead of gathering new data. For all of these new data sets a prediction is made and thereafter a final average of the new predictions is produced. This average prediction has lower variance compared to the single data set prediction but with similar bias. A delimitation is that the new data sets are correlated to the first one as well as to each other.

2.5 Random Forest

A way of getting past the correlation problem of bagging is random forest. In contrast to bagging which can be applied to any classification or regression problem, random forest is a way of reducing variance specifically for classification and regression trees. A central part of this method, also derived from its name, is to make the trees more random in an attempt to reduce the correlation. Instead of taking the different possible inputs (x_1,.....,x_p) into account as splitting variables for every split in a tree, a random set ($q \leq p$) of splitting variables are created for each split. Although this increases the variance in a separate tree, the average variance of the combined predictions is reduced.

3 Model Evaluation

3.1 Data Preprocessing and Feature Weights

Before the methods were applied to the data some processing of the data was made. First, the data was normalized, scaled from 0 to 1. Afterwards, both f test and Pearson correlation coefficient was used to find the features which were the most significant for the output data. Both methods selected the same features for each new dataframe when the dimensionality was reduced. When evaluating less than all features, the features are applied in order of highest significance. For example, when applying five features the five most significant features, 'acousticness', 'danceability', 'energy', 'loudness' and 'speechiness', are applied.

For example, the logistic regression model received the lowest misclassification error when the number of features in the X train data were reduced to five (see table 1 in 4. Model Results) with both the f test and linear correlations method, where the same features were selected.

One approach of creating a better correlation between the input and output data is by transforming the input data. Using a number of transformations, namely square product, square root, log10, exponential, sinus and cube product. After the data is transformed for every feature in every transformation, the highest linear correlation is decided. For example, the feature 'acousticness' had a higher linear correlation with the output feature 'label' when transformed with square root, whereas 'energy' had a higher linear correlation when transformed with square product. Even though there is a higher correlation, the absolute increase is never larger than 0.1. There is a slight difference in the order, changing loudness to the second most important feature instead of fourth.

3.2 Model Evaluation Metric

The models are evaluated using cross validation, splitting the training into 10 equal parts, creating training and validation data sets of 675 and 75 respectively, according to Lecture 5 of this course. The mean validation misclassification error was used to analyze each model's performance for the different number of features, applied in order of highest absolute linear correlation, which was found by taking the mean value for all iterations in the 10-fold cross validation.

This metric is used to compare and evaluate every model considered with 1 to 13 features, applied in order of significance, using both untransformed and transformed input data.

4 Model Results

The criterion for which method performed 'best' was the one with lowest misclassification error according to the subsection 3.2 Evaluation Metric. In Table 1 below, the lowest misclassification rates from using 1 to 13 of the training features is presented. The model with the lowest misclassification error rate is the Random forest model with 7 features and using transformed data. "In production", this model had an accuracy of 75%. This will be discussed in the following section (5. Discussion).

The parameters for the different methods were decided through evaluation of the misclassification error. In the kNN evaluation 16 neighbours resulted in a local minimum whilst 35 neighbours were inside a span of generally low misclassification error. The same reasoning decided the number of trees for random forest.

Model	**Number of features**	**Misclassification error**
Logistic regression	5	19.47%
Logistic regression (transformed data)	5	18.27%
LDA	5	18.40%
LDA (tranformed data)	3	17.87%
QDA	5	19.47%
QDA (transformed data)	3	17.73%
kNN, k = 16	2	18.00%
kNN, k = 16 (transformed data)	5	16.53%
kNN, k = 35	5	18.80%
kNN, k = 35 (transformed data)	8	17.73%
Bagging	8	17.47%
Bagging (transformed data)	9	17.87%
Random forest, n = 150	9	15.47%
Random forest, n = 150 (transformed data)	7	15.47%
Predicting ones	-	39.73%
Predicting ones (transformed data)	-	39.73%

Table 1: Misclassification errors for different evaluated models

5 Discussion

In different trials of putting models "in production", the group has found models with higher accuracy than the presented one of this report. These models had higher accuracy but also a higher misclassification error in the model evaluation. This qualifies the decided model evaluation method to be critiqued. For example, the accuracy of the bagging model with all features considered, using untransformed data, had a misclassification error of $\approx 17.47\%$ but an accuracy of 80%.

When applying the cross validation to measure misclassification, an "industry standard" of 10 folds was chosen. However, this does not have to statistically represent the final training and test data sets. Using 10 folds, training and validation data are split into groups of 675 and 75 data points each. The optimal split will have a large enough training and test size to estimate an E_{new} error. For the cross validation to accurately estimate this error, there needs to be a balance between a large enough training and test set to represent to final training and test sets. In this case, the industry standard of splitting into 10 subsets does not show in the results as an accurate representation of the evaluated models. However, it is not known if another amount of splits would create a better representation.

Another aspect to take into consideration is that the tested 200 songs are not the final E_{new}. The two-hundred songs decide the accuracy of the test, but in *true* production, more than the two-hundred songs are used. A model performing worse on the two-hundred songs does not mean it will perform worse on four-, six- or eight-hundred songs. With a different test data set, the cross-validation may be estimating the E_{new} correctly.

In the evaluation of the bias-variance trade off an interesting conclusion of our model selection can be made. Theoretically, the bagging model should have a higher variance compared to random forest models, since the accuracy in predicting unknown data decreased using the random forest model we could argue for the model to underfit. In the pursuit of lower misclassification errors the model complexity was often decreased since fewer features were taken into consideration. When observing the true accuracy, the lower accuracy implies an underfit model. This would suggest in a bias-variance trade off that the estimated E_{new} error has a minimum error at a lower model complexity than the actual E_{new} error. The cross-validation implemented favors a lower model complexity, which ultimately decreases the true accuracy. This results in an increased bias error instead of lowering the variance, as the intent was with lowering the model complexity by removing features. However, it is worth pointing out that some of the features have very low f-test or linear correlation scores, which would allow to doubt their importance in the final result.

The transformation of data was executed to further accentuate the correlation between some features and the label LIKE or DISLIKE. The transformation itself only had a small absolute percentage change on the correlation of the features and the label of the song, which makes one speculate about the necessity of the transformation. The data transformation resulted however in a lower misclassification error in the evaluation metric. In the project, a PCA transformation was evaluated. As there was no significant decrease of misclassification error, the group decided not to implement it.

Another data related procedure to highlight is the correlation evaluation. Since the assignment is a binary classification problem the output has either the value one or zero. Correlation with input features will not therefore not be linear. Both the f test and the Pearson correlation coefficient treats linear problems and when evaluating these two methods, the same features was categorically obtained. One could question the use of these test in regards to our formulated problem. Instead, a non linear or categorical method should have been applied in the analysis of how the different features are correlated. It is worth considering that even though the linear correlation is not a categorical significance test, it shows some kind of level of significance between the input feature and the output.

6 Conclusion

In summary, the chosen model evaluation method presented the model Random Forest, considering the seven most significant features and using transformed input data. This resulted in a production accuracy of 75%. Since the misclassification error rate in the training data does not correspond to the accuracy of the test data ”in production”, a discussion was raised about the following potential problems with the model evaluation method. The 10-fold cross validation measuring the misclassification error might not have been statistically representative for the given data set. The cross validation in order favoured a lower model complexity when looking at the misclassification rate, which in turn resulted in an increased bias error instead of lowering the variance. Finally, the data preparation process including transforming data and linear correlation evaluation was discussed. Since the label LIKE or DISKLIKE is a discrete classification output, checking for a linear correlation with the input features might not have contributed to making the model more accurate. Overall, the model evaluation method should be reevaluated.

7 Reflection Task

When designing a machine learning system it is important to reflect on how the system might affect people and society when the final version is in use. A ‘code of ethics’ is good to have in mind to make sure that the creation that is being designed can be ethically motivated. In this case, a machine learning system was to be designed for decision support to sales agents on an insurance company. That is a profession that handles large quantities of sensitive and very personal data about their customers. A machine learning system could streamline the workflow, giving the sales agent more time for other tasks that might not be prioritized otherwise. However, due to prejudices and cultural biases in the training data that forms the foundation of the system, discriminating societal structures might be amplified in the model. Therefore, it is necessary to evaluate the ethical aspect of the system and its implementation.

In the IEEE Code of Ethics (IEEE, 2018), one of the points recognizes the importance of treating all persons fairly and to not engage in acts of discrimination based on race, religion, gender, disability, age, national origin, sexual orientation, gender identity, or gender expression. Another point states an ethical obligation to be honest and realistic in stating claims or estimates based on available data. The two codes combined regards machine learning systems engineering and implies a social responsibility to not only consider the way that the system performs in terms of bias that can be harmful to the insurance company’s customers, but also a responsibility to inform the insurance company about these risks. The ethical obligation to be honest and realistic in the claims and estimations that the machine learning system creates implies informing and educating the client (the insurance company) about the risk of obtaining biases in the system. There is also an ethical responsibility to provide transparency to the insurance companies customers about how their data is being handled.

In addition to being careful not to discriminate, one of the other areas in the IEEE Code of Ethics considers improving the general understanding of the capabilities and implications of emerging technologies. The goal is to reach a higher level of understanding, for individuals and the society as a whole, in regards to the consequences of the implication of an intelligent system within the insurance business. The consequences of a machine learning bias could in the case of insurance greatly impact the financial situation of an individual, creating a systemic bias which would affect larger groups of society when generalizing people based on for example skin color or area of residence. This creates a clear disadvantage for certain people. Therefore, the increased general understanding of capabilities and implications of machine learning in decision making should be considered as of great importance. In this sense, educating the client is necessary and the least amount of action an engineer should take.

On the other hand, one could argue that engineers do not have the responsibility to educate and inform about possibly biased systems is that the system is designed to be less biased than humans are. If the system replaces a task humans traditionally perform, chances of the human having bias is quite high. This is something Debrusk points out in his article *The Risk of Machine-Learning Bias (and How to Prevent It)*, he states that *“...if humans are involved in decisions, bias always exists...”* (2018). If the system is tested and designed in such a way that biases are limited, or at least minimized, then one can argue that this would be a more objective process than a human taking the decision. We have to assume that the engineers that design the machine learning system that is to help decision making for an insurance company are up to speed about the problems around machine learning bias. Presumably, they would design the product as well as possible and test the system repeatedly before they give the system to the company. Then, if the engineers did their job right, the system would be optimized to have as little noticeable bias as possible, and ever perform better than humans would. In that case, the machine learning engineer could refer to the third principle in the Code of Honour of The Swedish Association of Graduate Engineers, stating that the engineer's aim is to provide knowledge that will reduce the risks within technologies (Sveriges Ingenjörer, 2018). The machine learning engineer could argue that he or she is following this principle and the requested system for the insurance company will provide the best basis for the insurance company's decision-making process. Thus, there is no need for the engineers to explain about biased claims or estimations, since all human claims and estimations are biased as well. Therefore there is no ethical responsibility for the engineers in that aspect.

In the reflection task text, the customer has not specified wanting an absolute un-biased system. If that were the case, a disclaimer from the engineers that ask is impossible could be in order. If not, it could be argued that the engineers do not have to burden the responsibility to inform and educate their employer or outsourcer about potential bias in the system. If the engineers have delivered what is specified in their contract and the client is satisfied with the result, then the engineers have fulfilled their part of the agreement. This is based on the assumption that the client has some basic knowledge about the technology that is requested for the company. In this case, the machine learning engineer could assume that the manager in the insurance company knows about the risks, such as bias, in the company's data when a machine learning system for decision support is requested. DeBrusk (2018) states that managers need to realise that bias exists in data sets and that they cannot view them as purely objective.

It lies in the direct interest of insurance companies to profile their customers. Based on age and gender, insurance will cost differently. In nature, insurance companies will carry out biased decision-making. Allowing a machine learning algorithm to carry out this task should not burden the engineers. As the environment, data and model is changing over time, engineers should only be responsible for maintaining and developing the model (DeBrusk, 2018). The machine learning engineers shouldn't need to feel responsibility to educate the clients, because the company - and the managers - should know about the requested technology (if it is not specified in the agreement). The engineers are only the deliverymen of the product, it is up to the end user to use it with caution.

If we are to include our personal thoughts on the matter, the group has discussed the ethical responsibility and code of profession that comes with machine learning engineering. To summarize, the arguments differ in being whether the engineer is producing the bias or the customers are responsible for the bias produced in the implementation. Either the engineer could be seen as just the deliveryman of a product, or taking on the responsibility of being the producer of the model.

One viewpoint is that designing a machine learning algorithm includes checking for biases, that it should be considered a part of the engineering design. When designing any machine learning model it is evaluated through variance and bias. Models with high variance or bias are not desirable therefore maintaining a low bias is part of the design process, something which goes for machine learning biases as well.

It could be argued that the demands should come from the client and if the client does not explicitly asks for unbiased models in certain aspects, then the assignment is fulfilled and it might be hard to blame the engineer for biases. However, personally, we believe that the engineer is responsible for explaining what it has produced in a transparent manner.

8 References

DeBrusk, Chris (Mar. 2018). The Risk of Machine-Learning Bias (and How to Prevent It). URL: https://sloanreview.mit.edu/article/the-risk-of-machine-learning-bias-and-how-to-prevent-it/.

IEEE (2018). IEEE Code of Ethics. URL: https://www.ieee.org/about/corporate/governance/p7-8.html.

MP_final

February 21, 2020

[79]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import sklearn.preprocessing as skl_pre
import sklearn.linear_model as skl_lm
import sklearn.discriminant_analysis as skl_da
import sklearn.neighbors as skl_nb
import sklearn.metrics as skl_met
import sklearn.model_selection as skl_ms
import sklearn.decomposition as skl_dec
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier

import seaborn as sns

from IPython.display import set_matplotlib_formats
set_matplotlib_formats('png')
from IPython.core.pylabtools import figsize
figsize(10, 6) # Width and hight
plt.style.use('seaborn-white')
```

[80]:
```
train_url = 'training_data.csv'
train_df = pd.read_csv(train_url, na_values='?', dtype={'ID': str}).dropna().
 ↪reset_index()

min_max_scaler = skl_pre.MinMaxScaler()
train_values = train_df.values #returns a numpy array
train_scaled = min_max_scaler.fit_transform(train_values)

train_df = pd.DataFrame(train_scaled,columns = list(train_df.columns.values)).
 ↪drop(columns = 'index') #drop index
train_df.head()
```

[80]:
```
   acousticness  danceability  duration    energy  instrumentalness       key  \
0      0.717303      0.463026  0.103325  0.519148          0.843847  0.727273
1      0.193158      0.690557  0.269951  0.613492          0.000000  0.363636
2      0.335009      0.594994  0.284262  0.452194          0.000004  0.454545
```

```
3  263  0.604627      0.799772  0.159891  0.214811          0.217166  0.454545
4       0.888330      0.407281  0.230079  0.456252          0.000179  0.545455

   liveness  loudness  mode  speechiness     tempo  time_signature   valence  \
0  0.092147  0.507981   0.0     0.030103  0.432113            0.75  0.116585
1  0.250262  0.779758   1.0     0.012185  0.459671            0.75  0.582714
2  0.107853  0.698741   1.0     0.008314  0.567220            0.75  0.176046
3  0.167539  0.639741   1.0     0.027953  0.365280            0.75  0.812062
4  0.047330  0.738406   0.0     0.016772  0.236229            0.75  0.270546

   label
0    1.0
1    1.0
2    1.0
3    1.0
4    1.0
```

[81]:
```python
X = train_df.drop(columns =['label'])
y = train_df['label']

n_fold = 10
```

[82]:
```python
#Choose best k. Same procedure was done with random forest, using K =
↪[0,10,20,40,60,80,100,130,150,180,200]
cv = skl_ms.KFold(n_splits = n_fold, random_state = 2, shuffle = True)
K = np.arange(1,200)
missclassification = np.zeros(len(K))
for train_index, val_index in cv.split(X):
    X_train, X_val = X.iloc[train_index], X.iloc[val_index]
    y_train, y_val = y.iloc[train_index], y.iloc[val_index]

    for j,k in enumerate(K):
        knn_model = skl_nb.KNeighborsClassifier(n_neighbors=k).
↪fit(X_train,y_train)
        y_pred = knn_model.predict(X_val)
        missclassification[j] += (np.mean(y_pred != y_val))

missclassification /= n_fold
plt.plot(K,missclassification)
plt.show()
#k around 30 gives a best result (30-40)
```

output_3_0.png

264

[83]:
```python
corr = train_df.corr()
print(corr)
#print(np.size(corr['label']))
corr_feat = corr.index
```

```
                  acousticness  danceability  duration    energy  \
acousticness          1.000000     -0.417974  0.054988 -0.781691
danceability         -0.417974      1.000000 -0.231120  0.360971
duration              0.054988     -0.231120  1.000000 -0.093435
energy               -0.781691      0.360971 -0.093435  1.000000
instrumentalness      0.331659     -0.238865  0.161803 -0.267846
key                  -0.065184      0.055302 -0.002089  0.066970
liveness             -0.140326     -0.115735 -0.002576  0.235887
loudness             -0.695163      0.396021 -0.179952  0.830081
mode                  0.111980     -0.058461 -0.011989 -0.102567
speechiness          -0.215614      0.272283 -0.110645  0.173371
tempo                -0.149472      0.064002 -0.052321  0.197741
time_signature       -0.205854      0.222486 -0.003030  0.241667
valence              -0.233485      0.483361 -0.256984  0.364495
label                 0.479307     -0.368501  0.138562 -0.459088

                  instrumentalness       key  liveness  loudness      mode  \
acousticness              0.331659 -0.065184 -0.140326 -0.695163  0.111980
danceability             -0.238865  0.055302 -0.115735  0.396021 -0.058461
duration                  0.161803 -0.002089 -0.002576 -0.179952 -0.011989
energy                   -0.267846  0.066970  0.235887  0.830081 -0.102567
instrumentalness          1.000000 -0.020726 -0.050720 -0.429529 -0.032180
key                      -0.020726  1.000000 -0.055710  0.009126 -0.158468
liveness                 -0.050720 -0.055710  1.000000  0.154176 -0.023585
loudness                 -0.429529  0.009126  0.154176  1.000000 -0.048111
mode                     -0.032180 -0.158468 -0.023585 -0.048111  1.000000
speechiness              -0.145104  0.081137  0.106747  0.188873 -0.118390
tempo                    -0.081531 -0.084476 -0.008093  0.188127  0.013911
time_signature           -0.024597  0.096863  0.037201  0.169130 -0.044799
valence                  -0.228774  0.068146  0.051110  0.294852  0.051939
label                     0.133523 -0.075631 -0.108682 -0.424345  0.080375

                  speechiness     tempo  time_signature   valence     label
acousticness        -0.215614 -0.149472       -0.205854 -0.233485  0.479307
danceability         0.272283  0.064002        0.222486  0.483361 -0.368501
duration            -0.110645 -0.052321       -0.003030 -0.256984  0.138562
energy               0.173371  0.197741        0.241667  0.364495 -0.459088
instrumentalness    -0.145104 -0.081531       -0.024597 -0.228774  0.133523
key                  0.081137 -0.084476        0.096863  0.068146 -0.075631
liveness             0.106747 -0.008093        0.037201  0.051110 -0.108682
```

```
loudness             0.188873  0.188127        0.169130  0.294852 -0.424345
mode                -0.118390  0.013911       -0.044799  0.051939  0.080375
speechiness          1.000000  0.139993        0.088062  0.101257 -0.480931
tempo                0.139993  1.000000        0.027999  0.076123 -0.071652
time_signature       0.088062  0.027999        1.000000  0.143921 -0.149962
valence              0.101257  0.076123        0.143921  1.000000 -0.178546
label               -0.480931 -0.071652       -0.149962 -0.178546  1.000000
```

[84]:
```
label_corr = corr['label'].drop(index = 'label')
variables = len(label_corr)
order = abs(label_corr).sort_values(ascending = False).index
```

[85]:
```
#highest linear correlation value
models = [skl_lm.LogisticRegression(solver = 'liblinear'), skl_nb.
 ↪KNeighborsClassifier(n_neighbors=16), skl_nb.
 ↪KNeighborsClassifier(n_neighbors=35), skl_da.LinearDiscriminantAnalysis(),␣
 ↪skl_da.QuadraticDiscriminantAnalysis()
, RandomForestClassifier(n_estimators = 150), BaggingClassifier(), '1:s']
model_labels = ['LogReg. ', 'kNN, k = 16. ', 'kNN, k = 35. ', 'LDA. ', 'QDA.',␣
 ↪'Random Forest, n = 150.','Bagging.', '1:s.']

for mi, model in enumerate(models):
    print('\n',model_labels[mi], '\n')
    model_label = model_labels[mi]
    missclassifications = []

    for i in range(1,len(order)+1):
        X_bf = X[order[0:i]] #bf = brute_force

        missclassification = 0
        for train_index, val_index in cv.split(X_bf):
            X_train, X_val = X_bf.iloc[train_index], X_bf.iloc[val_index]
            y_train, y_val = y.iloc[train_index], y.iloc[val_index]
            if not type(model) == str:
                model.fit(X_train,y_train)
                y_pred = model.predict(X_val)
            else:
                y_pred = np.array(np.repeat(1,len(X_val)))

            missclassification += (np.mean(y_pred != y_val))

        missclassification /= n_fold
        missclassifications.append(missclassification)
        print('Features: ', i, 'Missclassification error mean: ',␣
 ↪missclassification)
```

```
LogReg.
```

266

```
Features:  1 Missclassification error mean:  0.24666666666666667
Features:  2 Missclassification error mean:  0.22533333333333333
Features:  3 Missclassification error mean:  0.22133333333333333
Features:  4 Missclassification error mean:  0.21200000000000002
Features:  5 Missclassification error mean:  0.19466666666666668
Features:  6 Missclassification error mean:  0.20133333333333336
Features:  7 Missclassification error mean:  0.19733333333333333
Features:  8 Missclassification error mean:  0.19866666666666663
Features:  9 Missclassification error mean:  0.1973333333333333
Features:  10 Missclassification error mean:  0.19866666666666663
Features:  11 Missclassification error mean:  0.19466666666666668
Features:  12 Missclassification error mean:  0.19733333333333336
Features:  13 Missclassification error mean:  0.19333333333333336

 kNN, k = 16.

Features:  1 Missclassification error mean:  0.2693333333333333
Features:  2 Missclassification error mean:  0.18
Features:  3 Missclassification error mean:  0.1906666666666667
Features:  4 Missclassification error mean:  0.1853333333333333
Features:  5 Missclassification error mean:  0.18
Features:  6 Missclassification error mean:  0.196
Features:  7 Missclassification error mean:  0.1973333333333333
Features:  8 Missclassification error mean:  0.18133333333333337
Features:  9 Missclassification error mean:  0.184
Features:  10 Missclassification error mean:  0.18933333333333333
Features:  11 Missclassification error mean:  0.192
Features:  12 Missclassification error mean:  0.23866666666666667
Features:  13 Missclassification error mean:  0.2373333333333334

 kNN, k = 35.

Features:  1 Missclassification error mean:  0.26666666666666666
Features:  2 Missclassification error mean:  0.20266666666666663
Features:  3 Missclassification error mean:  0.2026666666666667
Features:  4 Missclassification error mean:  0.18933333333333333
Features:  5 Missclassification error mean:  0.188
Features:  6 Missclassification error mean:  0.19466666666666665
Features:  7 Missclassification error mean:  0.1973333333333333
Features:  8 Missclassification error mean:  0.19466666666666668
Features:  9 Missclassification error mean:  0.2013333333333333
Features:  10 Missclassification error mean:  0.192
Features:  11 Missclassification error mean:  0.1973333333333333
Features:  12 Missclassification error mean:  0.21466666666666664
Features:  13 Missclassification error mean:  0.22133333333333333

 LDA.
```

267

```
Features:  1 Missclassification error mean:  0.24266666666666667
Features:  2 Missclassification error mean:  0.196
Features:  3 Missclassification error mean:  0.19999999999999998
Features:  4 Missclassification error mean:  0.192
Features:  5 Missclassification error mean:  0.18400000000000002
Features:  6 Missclassification error mean:  0.19066666666666668
Features:  7 Missclassification error mean:  0.19466666666666665
Features:  8 Missclassification error mean:  0.19466666666666665
Features:  9 Missclassification error mean:  0.19066666666666668
Features:  10 Missclassification error mean:  0.19333333333333333
Features:  11 Missclassification error mean:  0.19466666666666668
Features:  12 Missclassification error mean:  0.188
Features:  13 Missclassification error mean:  0.19333333333333333
```

```
 QDA.
```

```
Features:  1 Missclassification error mean:  0.24533333333333332
Features:  2 Missclassification error mean:  0.22000000000000003
Features:  3 Missclassification error mean:  0.19866666666666666
Features:  4 Missclassification error mean:  0.21600000000000003
Features:  5 Missclassification error mean:  0.19466666666666665
Features:  6 Missclassification error mean:  0.196
Features:  7 Missclassification error mean:  0.21733333333333332
Features:  8 Missclassification error mean:  0.22933333333333333
Features:  9 Missclassification error mean:  0.24000000000000005
Features:  10 Missclassification error mean:  0.24533333333333332
Features:  11 Missclassification error mean:  0.24133333333333332
Features:  12 Missclassification error mean:  0.24
Features:  13 Missclassification error mean:  0.24266666666666664
```

```
 Random Forest, n = 150.
```

```
Features:  1 Missclassification error mean:  0.3186666666666667
Features:  2 Missclassification error mean:  0.19999999999999998
Features:  3 Missclassification error mean:  0.192
Features:  4 Missclassification error mean:  0.1653333333333333
Features:  5 Missclassification error mean:  0.15599999999999997
Features:  6 Missclassification error mean:  0.16133333333333333
Features:  7 Missclassification error mean:  0.16266666666666665
Features:  8 Missclassification error mean:  0.16133333333333336
Features:  9 Missclassification error mean:  0.16133333333333333
Features:  10 Missclassification error mean:  0.16799999999999998
Features:  11 Missclassification error mean:  0.16533333333333333
Features:  12 Missclassification error mean:  0.172
Features:  13 Missclassification error mean:  0.16666666666666669
```

```
 Bagging.
```

```
268
Features:  1 Missclassification error mean:  0.32133333333333336
Features:  2 Missclassification error mean:  0.22533333333333333
Features:  3 Missclassification error mean:  0.2013333333333333
Features:  4 Missclassification error mean:  0.20133333333333336
Features:  5 Missclassification error mean:  0.2
Features:  6 Missclassification error mean:  0.1986666666666667
Features:  7 Missclassification error mean:  0.1746666666666667
Features:  8 Missclassification error mean:  0.19333333333333336
Features:  9 Missclassification error mean:  0.17466666666666666
Features:  10 Missclassification error mean:  0.18799999999999997
Features:  11 Missclassification error mean:  0.184
Features:  12 Missclassification error mean:  0.1786666666666667
Features:  13 Missclassification error mean:  0.18266666666666664

 1:s.

Features:  1 Missclassification error mean:  0.39733333333333337
Features:  2 Missclassification error mean:  0.39733333333333337
Features:  3 Missclassification error mean:  0.39733333333333337
Features:  4 Missclassification error mean:  0.39733333333333337
Features:  5 Missclassification error mean:  0.39733333333333337
Features:  6 Missclassification error mean:  0.39733333333333337
Features:  7 Missclassification error mean:  0.39733333333333337
Features:  8 Missclassification error mean:  0.39733333333333337
Features:  9 Missclassification error mean:  0.39733333333333337
Features:  10 Missclassification error mean:  0.39733333333333337
Features:  11 Missclassification error mean:  0.39733333333333337
Features:  12 Missclassification error mean:  0.39733333333333337
Features:  13 Missclassification error mean:  0.39733333333333337
```

[86]:

```
#check if transformations create larger correlation value
transformations = [np.square, np.sqrt, lambda x: np.log10(x+1), np.exp, np.sin,␣
 ↪lambda x: np.power(x,3)]
correlations = pd.DataFrame(label_corr)#, columns =['0'])#, index =␣
 ↪list(label_corr.index))
correlations.columns = ['untouched']
max_correlations = correlations.copy()
#perform all transformations
for j,t in enumerate(transformations):
    transformed_df = train_df.transform(t)
    corr_t = transformed_df.corr()
    label_corr_t = corr_t['label'].drop(index = 'label')
    correlations[j] = label_corr_t
    #print(correlations)

#for every index decide highest correlation
```

269

```
best_corr = pd.Series()
best_corr_index = pd.Series()
for index in correlations.index:
    corr_comp = correlations.loc[index]
    order_corr = abs(corr_comp).sort_values(ascending = False).index
    max_corr_i = order_corr[0]
    max_corr = corr_comp[max_corr_i]
    #print(max_corr_i,max_corr)
    best_corr[index] = max_corr
    best_corr_index[index] = max_corr_i
    #print(best_corr)

max_correlations['best_corr'] = best_corr
max_correlations['transformation_index'] = best_corr_index
order_transformed = abs(max_correlations['best_corr']).sort_values(ascending = False).index

#print(max_correlations)

#create new df and new order
transf_df = train_df.copy() #change to test of used for final test
for index in correlations.index:
    row = max_correlations.loc[index]
    t_i = row['transformation_index']
    if(not t_i == 'untouched'):
        transf = transformations[t_i]
        col = transf_df[index].transform(transf)
        transf_df[index] = col
#out: transf_df, order_transformed
print('Transformed order:', '\n', order_transformed, '\nUntransformed order: \n', order)
```

```
Transformed order:
 Index(['speechiness', 'loudness', 'acousticness', 'energy', 'danceability',
       'valence', 'time_signature', 'instrumentalness', 'duration', 'liveness',
       'key', 'mode', 'tempo'],
      dtype='object')
Untransformed order:
 Index(['speechiness', 'acousticness', 'energy', 'loudness', 'danceability',
       'valence', 'time_signature', 'duration', 'instrumentalness', 'liveness',
       'mode', 'key', 'tempo'],
      dtype='object')
```

[87]:

```
X_trans = transf_df.drop(columns =['label'])
y_trans = transf_df['label']
```

```python
order_tf = order_transformed
model_labels = ['LogReg. Transformed Data.', 'kNN, k = 16. Transformed Data.', 'kNN, k = 35. Transformed Data.', 'LDA. Transformed Data.', 'QDA. Transformed Data.', 'Random Forest, n = 150. Transformed Data.','Bagging. Transformed Data.','1:s.']

for mi, model in enumerate(models):
    print('\n',model_labels[mi], '\n')
    model_label = model_labels[mi]
    missclassifications = []

    for i in range(1,len(order)+1):
        X_bf = X_trans[order_tf[0:i]] #bf = brute_force

        missclassification = 0
        for train_index, val_index in cv.split(X_bf):
            X_train, X_val = X_bf.iloc[train_index], X_bf.iloc[val_index]
            y_train, y_val = y.iloc[train_index], y.iloc[val_index]

            if not type(model) == str:
                model.fit(X_train,y_train)
                y_pred = model.predict(X_val)
            else:
                y_pred = np.array(np.repeat(1,len(X_val)))
            missclassification += (np.mean(y_pred != y_val))


        missclassification /= n_fold
        missclassifications.append(missclassification)
        print('Features: ', i, 'Missclassification error mean: ', missclassification)
```

```
 LogReg. Transformed Data.

Features:  1 Missclassification error mean:  0.244
Features:  2 Missclassification error mean:  0.19866666666666666
Features:  3 Missclassification error mean:  0.19333333333333333
Features:  4 Missclassification error mean:  0.20666666666666667
Features:  5 Missclassification error mean:  0.18266666666666667
Features:  6 Missclassification error mean:  0.18666666666666665
Features:  7 Missclassification error mean:  0.18666666666666665
Features:  8 Missclassification error mean:  0.18666666666666665
Features:  9 Missclassification error mean:  0.184
Features:  10 Missclassification error mean:  0.184
Features:  11 Missclassification error mean:  0.18533333333333335
Features:  12 Missclassification error mean:  0.18666666666666668
```

```
Features:  13 Missclassification error mean:  0.18666666666666668

 kNN, k = 16. Transformed Data.

Features:  1 Missclassification error mean:  0.26666666666666666
Features:  2 Missclassification error mean:  0.19066666666666668
Features:  3 Missclassification error mean:  0.172
Features:  4 Missclassification error mean:  0.16799999999999998
Features:  5 Missclassification error mean:  0.16533333333333333
Features:  6 Missclassification error mean:  0.18133333333333335
Features:  7 Missclassification error mean:  0.17866666666666667
Features:  8 Missclassification error mean:  0.176
Features:  9 Missclassification error mean:  0.17866666666666667
Features:  10 Missclassification error mean:  0.18000000000000005
Features:  11 Missclassification error mean:  0.19600000000000004
Features:  12 Missclassification error mean:  0.18666666666666668
Features:  13 Missclassification error mean:  0.19333333333333333

 kNN, k = 35. Transformed Data.

Features:  1 Missclassification error mean:  0.264
Features:  2 Missclassification error mean:  0.1866666666666667
Features:  3 Missclassification error mean:  0.18133333333333335
Features:  4 Missclassification error mean:  0.18933333333333333
Features:  5 Missclassification error mean:  0.17866666666666667
Features:  6 Missclassification error mean:  0.17333333333333334
Features:  7 Missclassification error mean:  0.17333333333333334
Features:  8 Missclassification error mean:  0.17733333333333332
Features:  9 Missclassification error mean:  0.17733333333333337
Features:  10 Missclassification error mean:  0.17866666666666667
Features:  11 Missclassification error mean:  0.19466666666666668
Features:  12 Missclassification error mean:  0.19333333333333333
Features:  13 Missclassification error mean:  0.20266666666666672

 LDA. Transformed Data.

Features:  1 Missclassification error mean:  0.24400000000000005
Features:  2 Missclassification error mean:  0.1933333333333333
Features:  3 Missclassification error mean:  0.17866666666666667
Features:  4 Missclassification error mean:  0.18266666666666667
Features:  5 Missclassification error mean:  0.18533333333333332
Features:  6 Missclassification error mean:  0.19066666666666668
Features:  7 Missclassification error mean:  0.18933333333333333
Features:  8 Missclassification error mean:  0.188
Features:  9 Missclassification error mean:  0.188
Features:  10 Missclassification error mean:  0.18800000000000003
Features:  11 Missclassification error mean:  0.18666666666666665
Features:  12 Missclassification error mean:  0.18666666666666662
```

```
Features:  13 Missclassification error mean:  0.18266666666666664

 QDA. Transformed Data.

Features:  1 Missclassification error mean:  0.24666666666666665
Features:  2 Missclassification error mean:  0.19333333333333333
Features:  3 Missclassification error mean:  0.17733333333333334
Features:  4 Missclassification error mean:  0.1746666666666667
Features:  5 Missclassification error mean:  0.184
Features:  6 Missclassification error mean:  0.18533333333333332
Features:  7 Missclassification error mean:  0.20133333333333336
Features:  8 Missclassification error mean:  0.192
Features:  9 Missclassification error mean:  0.20266666666666663
Features:  10 Missclassification error mean:  0.19866666666666666
Features:  11 Missclassification error mean:  0.2
Features:  12 Missclassification error mean:  0.21066666666666664
Features:  13 Missclassification error mean:  0.20533333333333333

 Random Forest, n = 150. Transformed Data.

Features:  1 Missclassification error mean:  0.32399999999999995
Features:  2 Missclassification error mean:  0.21200000000000002
Features:  3 Missclassification error mean:  0.172
Features:  4 Missclassification error mean:  0.168
Features:  5 Missclassification error mean:  0.15866666666666665
Features:  6 Missclassification error mean:  0.15866666666666668
Features:  7 Missclassification error mean:  0.15333333333333335
Features:  8 Missclassification error mean:  0.1653333333333333
Features:  9 Missclassification error mean:  0.156
Features:  10 Missclassification error mean:  0.16266666666666668
Features:  11 Missclassification error mean:  0.164
Features:  12 Missclassification error mean:  0.16799999999999998
Features:  13 Missclassification error mean:  0.16399999999999998

 Bagging. Transformed Data.

Features:  1 Missclassification error mean:  0.3133333333333333
Features:  2 Missclassification error mean:  0.22266666666666665
Features:  3 Missclassification error mean:  0.2066666666666667
Features:  4 Missclassification error mean:  0.19066666666666668
Features:  5 Missclassification error mean:  0.17866666666666664
Features:  6 Missclassification error mean:  0.18266666666666667
Features:  7 Missclassification error mean:  0.1746666666666667
Features:  8 Missclassification error mean:  0.18
Features:  9 Missclassification error mean:  0.18000000000000002
Features:  10 Missclassification error mean:  0.18133333333333335
Features:  11 Missclassification error mean:  0.184
Features:  12 Missclassification error mean:  0.19333333333333336
```

```
Features:  13 Missclassification error mean:  0.192

 1:s.

Features:  1 Missclassification error mean:  0.39733333333333337
Features:  2 Missclassification error mean:  0.39733333333333337
Features:  3 Missclassification error mean:  0.39733333333333337
Features:  4 Missclassification error mean:  0.39733333333333337
Features:  5 Missclassification error mean:  0.39733333333333337
Features:  6 Missclassification error mean:  0.39733333333333337
Features:  7 Missclassification error mean:  0.39733333333333337
Features:  8 Missclassification error mean:  0.39733333333333337
Features:  9 Missclassification error mean:  0.39733333333333337
Features:  10 Missclassification error mean:  0.39733333333333337
Features:  11 Missclassification error mean:  0.39733333333333337
Features:  12 Missclassification error mean:  0.39733333333333337
Features:  13 Missclassification error mean:  0.39733333333333337
```

[88]:

```python
#Choose best k. Same procedure was done with random forest, using K = [0,10,20,40,60,80,100,130,150,180,200]

cv = skl_ms.KFold(n_splits = n_fold, random_state = 2, shuffle = True)
K = np.arange(1,200)
missclassification = np.zeros(len(K))
for train_index, val_index in cv.split(X):
    X_train, X_val = X.iloc[train_index], X.iloc[val_index]
    y_train, y_val = y.iloc[train_index], y.iloc[val_index]

    for j,k in enumerate(K):
        knn_model = skl_nb.KNeighborsClassifier(n_neighbors=k).fit(X_train,y_train)
        y_pred = knn_model.predict(X_val)
        missclassification[j] += (np.mean(y_pred != y_val))

missclassification /= n_fold
plt.plot(K,missclassification)
plt.show()
#k around 30 gives a best result (30-40)
```

output_9_0.png

[ ]:
```python
#final train with variable choice
#print(lowest_missclassification,missc_index)
#k = missc_k
var = 11#missc_index + 1

test_url = 'songs_to_classify.csv'
test_df = pd.read_csv(test_url, na_values='?', dtype={'ID': str}).dropna().
 ↪reset_index()

test_values = test_df.values
test_scaled = min_max_scaler.fit_transform(test_values)

test_df = pd.DataFrame(test_scaled,columns = list(test_df.columns.values))#.
 ↪drop(columns = 'index')
#print(train_f_df.head())

X_test = test_df[order[0:var]]
#print(X_test_final.shape)
#print(X_test_final.head())

X_train = X[order[0:var]]
y_train = y

#knn_model = skl_nb.KNeighborsClassifier(n_neighbors=k).fit(X_train,y_train)
#y_pred = knn_model.predict(X_test)
#
model_bag = RandomForestClassifier(n_estimators = 150)#BaggingClassifier()
model_bag.fit(X_train,y_train)
y_pred = model_bag.predict(X_test)

ans = np.array2string(y_pred).replace('\n','').replace('.','').replace(' ','').
 ↪replace('[','').replace(']','')
ans_list = list(ans)
print(ans_list.count('0'), ans_list.count('1'))
print(ans)
```

[ ]:
```python
#final train with variable choice, transformed model/var

#print(lowest_missclassification,missc_index, missc_k)
#opt_k = missc_k

#opt_k = K[missc_k]
#print(lowest_missclassification,missc_index, opt_k)

opt_k = 35
var_amount = 8
#var_amount = missc_index + 1
```

275

```
var_index = order_transformed[0:var_amount]

test_url = 'songs_to_classify.csv'
test_df = pd.read_csv(test_url, na_values='?', dtype={'ID': str}).dropna().reset_index()

test_values = test_df.values
test_scaled = min_max_scaler.fit_transform(test_values)

test_df = pd.DataFrame(test_scaled,columns = list(test_df.columns.values))#.drop(columns = 'index')
#print(train_f_df.head())

X_test = test_df[var_index]
#print(X_test.head())
for index in var_index:
    row = max_correlations.loc[index]
    t_i = row['transformation_index']
    if(not t_i == 'untouched'):
        transf = transformations[t_i]
        col = X_test[index].transform(transf)
        X_test[index] = col
#print(X_test.head())

print(X_test.shape)
#print(X_test_final.head())

X_train = X_trans[var_index]
y_train = y

knn_model = skl_nb.KNeighborsClassifier(n_neighbors=opt_k).fit(X_train,y_train)
y_pred = knn_model.predict(X_test)

#model_rf = RandomForestClassifier(n_estimators = opt_k)
#model_rf.fit(X_train,y_train)
#y_pred = model_rf.predict(X_test)

ans = np.array2string(y_pred).replace('\n','').replace('.','').replace(' ','').replace('[','').replace(']','')
ans_list = list(ans)
print(ans_list.count('0'), ans_list.count('1'))
#print(y_pred,type(y_pred), np.array2string(y_pred).replace('. ','').replace('.\n','').replace('.','').replace(' ',''),len(np.array2string(y_pred)))
print(ans)
```