
Mini Project in Statistical Machine Learning

Spring 2021

Anonymous Author(s)

Affiliation

Address

email

Abstract

1 After testing three different algorithms on the data set we decided to proceed using
2 the Logistic Regression algorithm. It was enhanced using Ridge Regularization.
3 (Will add final result here later)

4 1 Introduction

5 The goal of the project was to determine whether it is possible to tell if the lead actor of a movie is
6 female or male, based on up to 13 different variables of data, and if so, what algorithm is best suited
7 for doing so. The variables in the data were: Number words female, Total words, Number of words
8 lead, Difference in words lead and co-lead, Number of male actors, Year, Number of female actors,
9 Number words male, Gross, Mean Age Male, Mean Age Female, Age Lead, Age Co-Lead.

10 1.1 Cross Validation

11 The Cross Validation method is a way of testing a classification method against another. The idea
12 behind cross validation is that the data set is split into different combinations of training and testing
13 data, and the model is tested on each one of them. This leads to a better understanding of the model's
14 performance as we basically can generate different training and testing data sets from the same set of
15 data.

16 2 Methods

17 The following methods were implemented and tested. The implementation of each method in Python
18 can be found in the appendix.

19 2.1 Linear Discriminant Analysis

20 The Linear Discriminant Analysis is a known method for dimensionality reduction, but can also
21 be very useful as a classification method. The goal of the method is to reduce the dimensions by
22 focusing on optimizing separating the classes. The method does this by calculating the separability
23 between its classes, known as the between-class variance, which is the distance between the mean of
24 its different classes [2]. This between-class variance can be calculated and stored in the so called
25 between-class matrix with the following equation [2],

$$S_b = \sum_{i=1}^g N_i (\bar{x}_i - \bar{x})(\bar{x}_i - \bar{x})^T \quad (1)$$

where N_i represents the sample size of class i , the \bar{x}_i is the sample mean of class i and the x_i is the overall mean. The next step in the method is to calculate the within-class variance, which is the distance between the mean and the sample of each class. This can be done with the following equation [2],

$$S_w = \sum_{i=1}^g (N_i - 1) S_i = \sum_{i=1}^g \sum_{j=1}^{N_i} (x_{i,j} - \bar{x})(\bar{x}_{i,j} - \bar{x})^T \quad (2)$$

where the parameters represent the same things as in the previous equation with the addition of $x_{i,j}$ as the sample of each class. The final step in the LDA method is to create a lower-dimensional space that maximizes the separability between the classes and at the same time minimizes variance within the classes. This can be done in several different ways, some examples are by using a least square method, by using the eigenvalues or by using singular value decomposition [2]. The Scikit we used in the project does this by using the singular value decomposition if nothing else is specified. This is preferred since it does not require the computation of a covariance matrix which makes it more efficient for higher dimensions.

The between-class variance (the distance between the mean of different classes) along with the within-class variance (the distance between the mean and the sample of each class) can be seen as measures of how difficult the separation will be. Where a shorter distance in the between-class variance and a longer distance in the within-class variance indicate a more difficult separation.

2.1.1 Evaluation

An important parameter when running the linear discriminant analysis is which solver to use. As we mentioned before the linear discriminant analysis can be run using the least square method, eigenvalues or singular value decomposition which we will henceforth refer to as lsqr, eigen and svd respectively. We used Scikits model-selection package to compare the different solvers. The package include methods which allow you to run repeated k-fold and grid-searches which run the solvers multiple times and then compare their averages. The best solver was svd with an accuracy of 0.865.

Another thing we explored with linear discriminant analysis was if adding a penalty in the form of a shrinkage would increase performance. This is only possible for the lsqr and eigen solvers since svd does not support this feature. The packages mentioned above includes options for testing shrinkage aswell and implementing and testing the shrinkage where the shrinkage varied between 0 and 1 with an increase of 0.01 in each step resulted in no enhancement in performance for the lsqr-solver but yielded a slight improvement for the eigen-solver when a shrinkage of 0.01 was used. However, the mean accuracy for the eigen-solver was still less than the mean accuracy for the scd performance and when taking these two tests in consideration we concluded that the svd-solver is the best one for our particular problem. We decided to proceed with the svd-solver when comparing the linear discriminant analysis with the other methods.

2.2 K-nearest neighbors

The k-nearest neighbors algorithm is a supervised classification algorithm that first was developed in the 1950's. It classifies an object based on the k closest objects from the training data set. The k is a positive, normally small integer that lets the model know how many neighbors the algorithm takes into consideration when classifying each object [?]. The choice of k is not clear and there is often data dependent. Different k values can result in different classification results, as seen in Figure 1 and Figure 2 below.

Since the KNN method relies on distance to assign class the data set is almost always normalized to handle different scaling and units in the data.

2.2.1 Evaluation

When testing for different k values from 1 to 23 on the entire normalized data set the KNN method failed to achieve an accuracy higher than in the low to mid seventies. This may seem like a good results, until the fact that the who data set is labeled 75.5 per cent male. The method's accuracy for

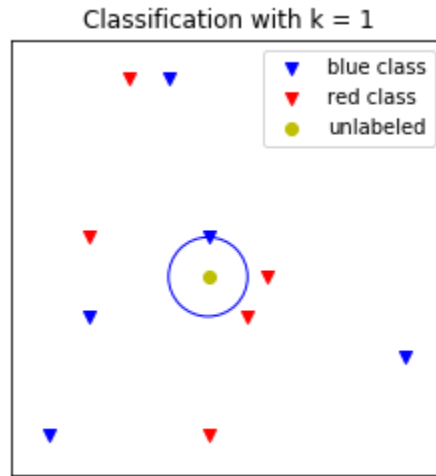


Figure 1: Classification of the yellow dot using $k = 1$ results in the blue class.

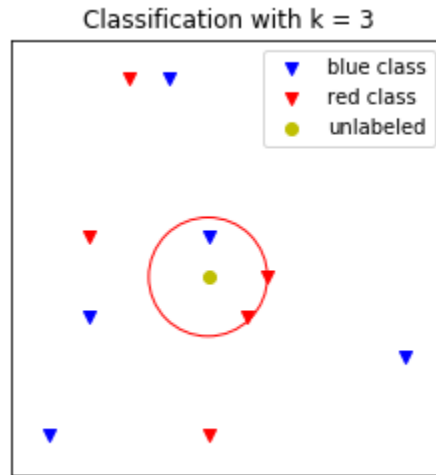


Figure 2: Classification of the yellow dot using $k = 3$ results in the red class.

73 predicting male actors was fairly good; ranging from around 0.85 to 0.95, but the number of female
 74 actors' accuracy predicted never exceeded 0.50.

75

76 As k increased the number of predicted female leads grew lower, this is a result of how skewed the
 77 data was towards male actors. For example: for a testing data set of size 159 and $k = 1$ the model
 78 predicted 60 female leads (38 % of the total actors), but for $k = 11$ on the same data set only 26 fe-
 79 male leads were predicted (16 %). In Table 1 the accuracy and number of predicted females are shown.

80

81 When using $k = 3$ and only one of the features in the data set the highest accuracy was achieved when
 82 only looking at the number of female actors. The model achieved an accuracy of 0.714, however the
 83 model only predicted that 18 of 259 lead actors were female, the lowest out of any single feature. The
 84 features Words Spoken by Males & Females, Year of Release and Gross resulted accuracies of 0.617,

Table 1: K-NN results for different K values.

K Value	Accuracy	Predicted Females
1	0.745	60
3	0.733	53
5	0.722	34
7	0.722	36
9	0.737	30
11	0.737	26
13	0.729	20
15	0.729	14
17	0.725	13
19	0.714	8
21	0.722	6
23	0.718	5

0.644, 0.629, each run predicting 53, 26, 46 female leads. When applying ten-fold sampling without replacement to the *KNN* model returned a lower accuracy and the number of predicted female leads was heavily reduced. We decided not to move forward with the *KNN* algorithm.

2.3 Logistic Regression

The following section uses content from the course literature [1], in order to explain the logistic regression model as a classifier.

In general, a classification model can be specified in terms of the conditional class probabilities

$$p(y = m|\mathbf{x}) \quad \text{for } m \in 1, \dots, M \quad (3)$$

which describes the probability for class $y = 1, \dots, M$, given some known input variable(s) \mathbf{x} . For binary classification problems, that is $M = 2$ and $y \in \{-1, 1\}$, we can learn a model $f(\mathbf{x})$ that describes the conditional class probability for the positive class $y = 1$

$$f(\mathbf{x}) = p(y = 1|\mathbf{x}) \quad (4)$$

where $0 \leq f(\mathbf{x}) \leq 1$, since it is a model for probability. And given by the law of total probability, we can model the conditional class probability for the negative class $y = -1$ as

$$p(y = -1|\mathbf{x}) = 1 - f(\mathbf{x}) \quad (5)$$

One classification method using conditional probabilities is the *logistic regression model*. The idea of the logistic regression is based on the *linear regression model*, together with conditional probabilities for classification. The linear regression model is given by

$$z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n + \epsilon = \boldsymbol{\theta}^T \mathbf{x} + \epsilon \quad (6)$$

where the output variable z , a numerical value, is assumed to be a linear combination of the input variables \mathbf{x} and unknown model parameters $\boldsymbol{\theta}$, including the intercept term θ_0 . The ϵ term accounts for random errors in the data, not explained by the model. The values assigned to the parameters $\boldsymbol{\theta}$ determine the relationship between the input and output variables described by the model.

In order to interpret the output z in terms of conditional class probabilities, the *logistic function* $h(z) = \frac{e^z}{1+e^z}$ is used, as it is restricted to the interval $[0,1]$. For binary classification problems, $y \in \{-1, 1\}$, this results in the *logistic regression model*

$$p(y = 1|\mathbf{x}; \boldsymbol{\theta}) = f(\mathbf{x}; \boldsymbol{\theta}) = \frac{e^{\boldsymbol{\theta}^T \mathbf{x}}}{1 + e^{\boldsymbol{\theta}^T \mathbf{x}}} \quad (7)$$

describing the conditional class probability for the positive class $y = 1$. Again, by the law of total probability, this also gives the logistic regression model for the negative class $y = -1$

$$p(y = -1|\mathbf{x}; \boldsymbol{\theta}) = 1 - f(\mathbf{x}; \boldsymbol{\theta}) = 1 - \frac{e^{\boldsymbol{\theta}^T \mathbf{x}}}{1 + e^{\boldsymbol{\theta}^T \mathbf{x}}} = \frac{e^{-\boldsymbol{\theta}^T \mathbf{x}}}{1 + e^{-\boldsymbol{\theta}^T \mathbf{x}}} \quad (8)$$

110 Note that there is no random error ϵ included the logistic regression model (7), as in the linear
 111 regression model (6), due to the randomness in classification that is statistically modeled by the
 112 conditional class probability $p(y|\mathbf{x})$, instead of an additional error.

113 So, by using the logistic function, the linear regression model - a model for solving regression
 114 problems - is modified into the logistic regression model - a model for classification problems. In
 115 order to use the logistic regression model for predicting class probabilities, or actual class predictions,
 116 we need to learn the parameters θ from the training data $T = \{\mathbf{x}_i, y_i\}_{i=1}^n$. The parameters θ can be
 117 learned by using the *likelihood function* $\ell(\theta)$, considering the (natural) logarithm of the likelihood
 118 function for numerical reasons,

$$\ln \ell(\theta) = \sum_{i=1}^n \ln p(y_i | \mathbf{x}_i; \theta) \quad (9)$$

119 to find the *maximum likelihood estimate* $\hat{\theta}$ for θ

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} p(\mathbf{y} | \mathbf{x}; \theta) = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^n \ln p(y_i | \mathbf{x}_i; \theta) \quad (10)$$

120 Solving the problem, means estimating, or learning, the parameters $\hat{\theta}$ that maximizes the likelihood
 121 that an input \mathbf{x}_i belongs to class y_i . Generally, this maximization problem is turned into the dual
 122 minimization problem $\underset{\theta}{\operatorname{argmin}} - \sum_{i=1}^n \ln p(y_i | \mathbf{x}_i; \theta)$, together with the statistical average, giving
 123 us $J(\theta) = -\frac{1}{n} \sum_{i=1}^n \ln p(y_i | \mathbf{x}_i; \theta)$, commonly known as the *cost function*. From the logistic
 124 regression model given in (7) and (8), together with the cost function $J(\theta)$, this gives us

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n \begin{cases} -\ln f(\mathbf{x}_i; \theta) & \text{if } y_i = 1 \\ -\ln (1 - f(\mathbf{x}_i; \theta)) & \text{if } y_i = -1 \end{cases} \quad (11)$$

125 which can be written in a more compact way, using the binary class formulation of $y_i = \{-1, 1\}$,
 126 since the logistic regression model in (7) and (8) gives us

$$\begin{cases} f(\mathbf{x}; \theta) = \frac{e^{\theta^T \mathbf{x}}}{1 + e^{\theta^T \mathbf{x}}} = \frac{e^{y_i \theta^T \mathbf{x}}}{1 + e^{y_i \theta^T \mathbf{x}}} & \text{if } y_i = 1 \\ 1 - f(\mathbf{x}; \theta) = \frac{e^{-\theta^T \mathbf{x}}}{1 + e^{-\theta^T \mathbf{x}}} = \frac{e^{y_i \theta^T \mathbf{x}}}{1 + e^{y_i \theta^T \mathbf{x}}} & \text{if } y_i = -1 \end{cases} \quad (12)$$

127 the same expression on both cases, and thus we can formulate the cost function (11) as

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n -\ln \frac{e^{y_i \theta^T \mathbf{x}}}{1 + e^{y_i \theta^T \mathbf{x}}} = \frac{1}{n} \sum_{i=1}^n -\ln \frac{1}{1 + e^{-y_i \theta^T \mathbf{x}}} = \frac{1}{n} \sum_{i=1}^n \ln(1 + e^{-y_i \theta^T \mathbf{x}}) \quad (13)$$

128 Hence, learning a logistic regression model, means solving

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n \ln(1 + e^{-y_i \theta^T \mathbf{x}}) \quad (14)$$

129 and this is equivalent to finding the maximum likelihood estimate $\hat{\theta}$ as written in (10). This is a
 130 nonlinear optimization problem that needs to be solved numerically, as there is no direct solution for
 131 finding $\hat{\theta}$.

132 Once we have learned the parameters $\hat{\theta}$, we can use the logistic regression model, (7) and (8), to
 133 predict class probabilities, given some test input data \mathbf{x}_* . The next step is to turn these predicted class
 134 probabilities into actual class predictions $\hat{\mathbf{y}}(\mathbf{x}_*) = 1$ or $\hat{\mathbf{y}}(\mathbf{x}_*) = -1$. The most common approach is
 135 to let $\hat{\mathbf{y}}(\mathbf{x}_*)$ be the most probable class by

$$\hat{\mathbf{y}}(\mathbf{x}_*) = \begin{cases} 1 & \text{if } f(\mathbf{x}; \hat{\theta}) > r, \\ -1 & \text{otherwise} \end{cases} \quad (15)$$

136 with decision threshold $r = 0.5$, but in general the threshold is chosen by the user $0 \leq r \leq 1$. The
 137 input points where the prediction changes from one class to the other, that is either of the two classes

138 ($y = 1$ or -1) is equally probable, is the decision boundary. This corresponds to solving the equation
 139

$$f(\mathbf{x}; \boldsymbol{\theta}) = 1 - f(\mathbf{x}; \boldsymbol{\theta}) \quad (16)$$

140 and for logistic regression this means

$$\frac{e^{\boldsymbol{\theta}^T \mathbf{x}}}{1 + e^{\boldsymbol{\theta}^T \mathbf{x}}} = 1 - \frac{e^{\boldsymbol{\theta}^T \mathbf{x}}}{1 + e^{\boldsymbol{\theta}^T \mathbf{x}}} \Leftrightarrow e^{\boldsymbol{\theta}^T \mathbf{x}} = 1 \Leftrightarrow \boldsymbol{\theta}^T \mathbf{x} = 0 \quad (17)$$

141 The equation $\boldsymbol{\theta}^T \mathbf{x} = 0$ indicates a (linear) hyperplane, which means that the decision boundary for
 142 logistic regression is always linear.

143 2.4 Lasso and Ridge Regularization

144 To improve a regression model and avoid overfitting we can use regularization. The key concept
 145 behind regularization is that we want to make sure that our parameters $\hat{\boldsymbol{\theta}}$ as small as possible. This is
 146 done to ensure that the model complexity won't be too high and can be done in many ways. In Ridge,
 147 or L^2 Regularization we alter the cost function by adding a penalty, defined as the sum of the square
 148 of the θ terms. They are then scaled by a penalty term, c (often called λ). The size of c can vary, but as
 149 it approaches ∞ θ will approach zero. In LASSO, or L^1 Regularization, the penalty is defined as the
 150 absolute value of the θ term. The difference between these methods is that in Ridge Regularization
 151 the model will typically push its θ parameters towards small non-zero values, whereas in LASSO
 152 Regularization the model will set θ values to zero, and therefore can act as a feature selection model.
 153 [1]

154 2.4.1 Evaluation

155 When plotting the validation errors from cross validation (e.i misclassification errors) in a box plot,
 156 Ridge Regularization out-performed both LASSO, as well as logistic regression without regularization,
 157 for our classification problem. Next step was to choose an appropriate value for the penalty term c .
 158 This was done by testing a myriad of values between 1 and 2000 and comparing the validation errors
 159 between these different models. Using cross validation ($k = 10$), the validation errors were calculated
 160 and plotted in a box diagram. We were looking for a value with an error as small as possible, but also
 161 with a small error variance too. Of the tested values of c , many produced similar mean errors, but
 162 $c = 1000$ had the "smallest box", i.e. the lowest error variance. c was therefore set to 1000.

163 3 Our Choice of Method to Proceed With

164 After testing the three aforementioned methods, we decided to proceed with the Logistic Regression
 165 method, and try to tune it. This was done since the model performed best in the cross validation
 166 testing.

167 4 Conclusions

168 When using the logistic regression method with Ridge regularization, $c = 1000$, we managed to
 169 achieve a validation error of between 0.08 and 0.18.

170 5 The Feature Importance Task

171 Unfortunately, we have not yet succeeded in evaluating the importance of the different features of the
 172 data. So far, we have examined the parameters $\hat{\boldsymbol{\theta}}$ of the logistic regression model with regularization,
 173 both of the Ridge and LASSO regression, since the values of these parameters indicate which input
 174 variables \mathbf{x} that are most important (i.e those that are most predictive). The input variables with the
 175 highest valued parameters, that is the ones that indicate importance, were especially *Number of male*
 176 *actors*, *Number of female actors*, followed by *Mean Age Male*. However, this is not intuitive to us, and
 177 we will go on and examine the feature importance by the following actions; choosing a combination
 178 of all input variables, including non-linear transformation of these, and measuring the AIC (Akaike
 179 Information Criterion), which will indicate the best fit.

180 **References**

- 181 [1] Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, and Thomas B. Schön. *Supervised*
182 *Machine Learning*, 2020.
183 <https://smlbook.org>
- 184 [2] C.R. Rao, Venu Govindaraju. *Handbook of Statistics, Machine Learning: Theory and Applica-*
185 *tions*, 2013

186 6 Appendix

```

187 1 import pandas as pd
188 2 import numpy as np
189 3 import matplotlib.pyplot as plt
190 4
191 5 import sklearn.preprocessing as skl_pre
192 6 import sklearn.linear_model as skl_lm
193 7 import sklearn.discriminant_analysis as skl_da
194 8 import sklearn.neighbors as skl_nb
195 9 import sklearn.model_selection as skl_ms
196 0
197 11 import os
198 12
199 13 #from IPython.display import set_matplotlib_formats
200 14 #set_matplotlib_formats('png')
201 15 from IPython.core.pylabtools import figsize
202 16 figsize(10, 6) # Width and height
203 17 #plt.style.use('seaborn-white')
204 18
205 19
206 20 # In[64]:
207 21
208 22
209 23 # Set working directory and read in the csv file
210 24 os.chdir("D:\Skola\System i teknik och samh lle\Statistisk
211      Maskininl rning\Project")
212 25 cwd = os.getcwd()
213 26 print("The current working directory is: ", cwd)
214 27
215 28 trainData = pd.read_csv("train.csv")
216 29
217 30
218 31 # In[65]:
219 32
220 33
221 34 #Make sure we can access the file and what it contains
222 35 trainData.info()
223 36 trainData.describe()
224 37 print(trainData)
225 38 pd.plotting.scatter_matrix(trainData)
226 39
227 40
228 41
229 42 # In[66]:
230 43
231 44
232 45 print(f"auto.shape: {trainData.shape}") #No. of rows, No. of columns
233 46
234 47 #Split the data randomly into a training test and a test set of
235      approximately same size
236 48 #Set seed to get reproducible results
237 49 np.random.seed(1)
238 50
239 51 trainI = np.random.choice(trainData.shape[0], size = 500, replace =
240      False)
241 52 trainIndex = trainData.index.isin(trainI)
242 53 train = trainData.iloc[trainIndex]
243 54 test = trainData.iloc[~trainIndex]
244 55
245 56 #Size of Data ---> 1039
246 57
247 58
248 59 # In[67]:
249 60

```

```

25061
25162 #Set up train and test data
25263
25364 X_train = train[['Number words female', 'Total words', 'Number of
254         words lead', 'Difference in words lead and co-lead',
25565         'Number of male actors', 'Year', 'Number of female
256         actors', 'Number words male', 'Gross',
25766         'Mean Age Male', 'Mean Age Female', 'Age Lead', 'Age
258         Co-Lead']]
25967 Y_train = train['Lead']
26068 X_test = test[['Number words female', 'Total words', 'Number of words
261         lead', 'Difference in words lead and co-lead',
26269         'Number of male actors', 'Year', 'Number of female
263         actors', 'Number words male', 'Gross',
26470         'Mean Age Male', 'Mean Age Female', 'Age Lead', 'Age Co
265         -Lead']]
26671 Y_test = test['Lead']
26772
26873
26974 # In[68]:
27075
27176
27277 # set up the KNN-solver for comparison
27378
27479 model = skl_nb.KNeighborsClassifier(n_neighbors = 1)
27580 model.fit(X_train, Y_train)
27681
27782 prediction = model.predict(X_test)
27883 print('Confusion Matrix :\n')
27984 print(pd.crosstab(prediction, Y_test), '\n')
28085 print(f'Accuracy: {np.mean(prediction == Y_test):.3f}')
28186
28287
28388 # In[69]:
28489
28590
28691 #Predict using LDA
28792
28893 model = skl_da.LinearDiscriminantAnalysis()
28994 model.fit(X_train, Y_train)
29095
29196 predict_prob = model.predict_proba(X_test)
29297 print('The class order in the model:')
29398 print(model.classes_)
29499
29500 print('Examples of predicted probabilities for the above classes:')
29601 with np.printoptions(suppress = True, precision = 4): #Supress
297         scientific notation, e.g. 1.0e-2.
29802     print(predict_prob[0:5], '\n\n Actual class:\n', trainData.loc
299         [0:5, 'Lead']) #Inspect the first five predictions
30003
30104
30205
30306 # In[70]:
30407
30508
30609 prediction = np.empty(len(X_test), dtype = object)
30710 prediction = np.where(predict_prob[:,0]>=0.5, 'Female', 'Male')
30811 print('First five predictions:')
30912 print(prediction[0:5], '\n') #Inspect the first five predictions after
310         Labeling
31113 print('First five actual data:')
31214 print(trainData.loc[0:5, 'Lead'])
31315
31416 #Confusion matrixt

```

```

3157 print(' \n Confusion matrix:')
3168 print(pd.crosstab(prediction, Y_test, ), '\n')
3179
3180 #Accuracy
3191 print(f'Accuracy: {np.mean(prediction == Y_test):.3f}')
```

32022

32123

3224 # In[71]:

3235

3246

3257 #Code for comparison of the different solvers for LDA

32628

32729 X_train2 = trainData[['Number words female', 'Total words', 'Number of
328 words lead', 'Difference in words lead and co-lead',
32930 'Number of male actors', 'Year', 'Number of female
330 actors', 'Number words male', 'Gross',
33131 'Mean Age Male', 'Mean Age Female', 'Age Lead', 'Age
332 Co-Lead']]

3332 Y_train2 = trainData['Lead']

33433

33534 # set up the model

33635 model = skl_da.LinearDiscriminantAnalysis()

33736 # setup the evaluation method

33837 cv = skl_ms.RepeatedStratifiedKFold(n_splits=10, n_repeats=3,
339 random_state=1)

34038 # set up grid

34139 grid = dict()

34240 grid['solver'] = ['svd', 'lsqr', 'eigen']

34341 # set up search

34442 search = skl_ms.GridSearchCV(model, grid, scoring='accuracy', cv=cv,
345 n_jobs=-1)

34643 # execute the search

34744 results = search.fit(X_train2, Y_train2)

34845 # summarize and evaluate which solver works best, we continue with the
349 best one

35046 print('Mean Accuracy: %.3f' % results.best_score_)

35147 print('Config: %s' % results.best_params_)

35248 print('best param: %s' % results.best_params_)

35349

35450

35551 # In[72]:

35652

35753

35854 #Check how shrinkage affects the methods

35955

36056 X_train3 = trainData[['Number words female', 'Total words', 'Number of
361 words lead', 'Difference in words lead and co-lead',
36257 'Number of male actors', 'Year', 'Number of female
363 actors', 'Number words male', 'Gross',
36458 'Mean Age Male', 'Mean Age Female', 'Age Lead', 'Age
365 Co-Lead']]

36659 Y_train3 = trainData['Lead']

36760

36861

36962 # set up the model

37063 model = skl_da.LinearDiscriminantAnalysis(solver='eigen')

37164 # set up the evaluation method

37265 cv = skl_ms.RepeatedStratifiedKFold(n_splits=10, n_repeats=3,
373 random_state=1)

37466 # set up the grid

37567 grid = dict()

37668 grid['shrinkage'] = np.arange(0, 1, 0.01)

37769 # set up the search

37870 search = skl_ms.GridSearchCV(model, grid, scoring='accuracy', cv=cv,
379 n_jobs=-1)

```

38071 # execute the search
38172 results = search.fit(X_train3, Y_train3)
38273 # summarize the affects of the shrinkage
38374 print('Mean Accuracy: %.3f' % results.best_score_)
38475 print('Config: %s' % results.best_params_)
38576
38677
38778 # In[73]:
38879
38980
39081 #cross-validation comparisons of the different methods
39182
39283 X_train4 = trainData[['Number words female', 'Total words', 'Number of
393 words lead', 'Difference in words lead and co-lead',
394 'Number of male actors', 'Year', 'Number of female
395 actors', 'Number words male', 'Gross',
396 'Mean Age Male', 'Mean Age Female', 'Age Lead', 'Age
397 Co-Lead']]
39886 Y_train4 = trainData['Lead']
39987
40088 n_fold = 10
40189
40290 models = []
40391 models.append(skl_da.LinearDiscriminantAnalysis()) #Set up LDA
40492 models.append(skl_nb.KNeighborsClassifier(n_neighbors = 3)) #Set up K-
405 nn with k = 2
40693 models.append(skl_lm.LogisticRegression(solver='liblinear')) #Set up
407 Logistic Regression
40894
40995 missclassification = np.zeros((n_fold, len(models)))
41096 cv = skl_ms.KFold(n_splits=n_fold, random_state = 1, shuffle = True)
41197
41298 for i, (train_index, val_index) in enumerate(cv.split(X_train4)):
41399 X_trainfold, X_val = X_train4.iloc[train_index], X_train4.iloc[
414 val_index]
41510 Y_trainfold, Y_val = Y_train4.iloc[train_index], Y_train4.iloc[
416 val_index]
41711
41812 for m in range(np.shape(models)[0]): #try different models
41913 model = models[m]
42014 model.fit(X_trainfold, Y_trainfold)
42115 prediction = model.predict(X_val)
42216 missclassification[i,m] = np.mean(prediction != Y_val)
42317 #print(missclassification[i,m])
42418
42519 modelLDA = models[0]
42620 X_lda = modelLDA.fit_transform(X_trainfold, Y_trainfold)
42721 #print(modelLDA.explained_variance_ratio_)
42822
42923
43024 print(missclassification)
43125 plt.boxplot(missclassification)
43226 plt.title('cross validation error for different methods')
43327 plt.xticks(np.arange(4)+1, ('LDA', 'K-nn', 'Logistic Regression'))
43428 plt.ylabel('validation error')
43529 plt.show()
43630
43721 #model.explained_variance_ratio_

438 1 #!/usr/bin/env python
439 2 # coding: utf-8
440 3
441 4 # Load info from the csv file train.csv, located at 'C:\Users\vikin\
442 Documents\STS\Statistisk maskininl rning\projects\1\train.csv'.
443 The file consists of 13 datapoints and then the classifier.

```

```

444 5 #
445 6 # Implement a KNN algorithm and use the training data as a basis for
446   it.
447 7 #
448 8 #
449 9 # Number of males:
450 10 # 785
451 11 #
452 12 # Number of females:
453 13 # 254
454 14 #
455 15 # -> predicting only males will result in an accuracy of
456   0.7555341674687199
457 16
458 17 # In[11]:
459 18
460 19
461 20 import matplotlib.pyplot as plt
462 21 import numpy as np
463 22 import csv
464 23
465 24 # KNN related functions
466 25
467 26 def euclidian_distance(p1, p2):
468 27     '''returns the euclidian distance between the two tuples p1 and p2
469   28     '''
470 29     d = 0
471 30     for i in range(len(p1)):
472 31         d += (p1[i] - p2[i])**2
473 32     return d**0.5
474 33
475 34 def knn_classify_me(k, distance_and_class):
476 35     '''returns the most common class from k points with shortest
477   36     distance
478 37         in the unordered list of tuples (distance, class) in
479   38     distance_and_class'''
480 39     distance_and_class.sort(key=lambda x: x[0])
481 40
482 41     class_dict = {}
483 42
484 43     for i in range(k):
485 44         temp_class = distance_and_class[i][1]
486 45         if temp_class in class_dict:
487 46             class_dict[temp_class] += 1
488 47         else:
489 48             class_dict[temp_class] = 1
490 49
491 50     return max(class_dict, key = class_dict.get)
492 51
493 52 def knn_return_distance_and_class_vector(p, training_values,
494   training_classes):
495 53     '''returns a list of tuples (distance, class) distance_and_class
496   54     for every instance in the two training lists and their
497   55     distance to the point p'''
498 56     distance_and_class = []
499 57
500 58     for i in range(len(training_values)):
501 59         current_point = training_values[i]
502 60         distance = euclidian_distance(p, current_point)
503 61         current_class = training_classes[i]
504 62         distance_and_class.append((distance, current_class))
505 63
506 64     return distance_and_class
507 65
508 66 def knn_p(k, point, training_values, training_classes):

```

```

5093     '''returns the class of point p according to the k nearest
510     neighbors
5114     in training_values and their classes in training_classes'''
5125     distance_and_class = knn_return_distance_and_class_vector(point,
513     training_values, training_classes)
5146     return knn_classify_me(k, distance_and_class)
5157
5168 def knn(k, testing_values, training_values, training_classes):
5179     '''returns a list of classes our testing_values get based on
5180     our training_values and training_classes'''
5191     classes = []
5202
5213     for point in testing_values:
5224         current_class = knn_p(k, point, training_values,
523         training_classes)
5245         classes.append(current_class)
5256     return classes
5267
5278
5289 # In[72]:
5290
5301
5312 # general data related functions
5323
5334
5345 def feasible(k, testing_values, training_values, training_classes):
5356     '''ensures that the length of all lists and tuples match'''
5367     if len(training_values) != len(training_classes):
5378         print("training data doesn't match")
5389         return False
5390
5401     if len(training_values[0]) != len(testing_values[0]):
5412         print("values don't match")
5423         return False
5434     if k > len(training_values):
5445         print("k is too big")
5456         return False
5467     return True
5478
5489 def results(predicted, actual):
5490     '''returns accuracy and prints all sorts of stuff'''
5501     predicted_males_correct = 0
5512     predicted_females_correct = 0
5523
5534     predicted_males_false = 0
5545     predicted_females_false = 0
5556
5567     for i in range(len(predicted)):
5578         if predicted[i] == "Male":
5589             if actual[i] == "Male":
5590                 predicted_males_correct += 1
5601             else:
5612                 predicted_males_false += 1
5623         else:
5634             if actual[i] == "Female":
5645                 predicted_females_correct += 1
5656             else:
5667                 predicted_females_false += 1
5678     print('total predicted males {}'.format(predicted_males_correct +
568     predicted_males_false))
5699     print('total predicted females {}'.format(
570     predicted_females_correct + predicted_females_false))
5710     print('predicted_males_correct / total males {}'.format(
572     predicted_males_correct/(predicted_males_correct+
573     predicted_females_false)))

```

```

57421     print('predicted_females_correct / total females {}'.format(
575         predicted_females_correct/(predicted_females_correct+
576         predicted_males_false)))
57722     return (predicted_males_correct + predicted_females_correct) / len
578         (predicted)
57923
58024 def normalize(eg_array):
58125     '''subtracts the mean and divides by the standard deviation'''
58226     return (eg_array - eg_array.mean(axis=0)) / eg_array.std(axis=0)
58327
58428 def split_into_k(arr, k = 10):
58529     '''returns a list of k subsections of arr'''
58630     part = len(arr) // k
58731     list_of_matrices = [0] * k
58832
58933     for i in range(k):
59034         if i == k - 1:
59135             list_of_matrices[i] = arr[part * i : ]
59236         else:
59337             list_of_matrices[i] = arr[part * i : part * (i + 1)]
59438     return list_of_matrices
59539
59640
59741 def combine_results(results_arrays):
59842     '''returns a list of the most common results for each index of the
599         results_arrays'''
60043     results = []
60144     for i in range(len(results_arrays[0])):
60245         temp = {}
60346         for j in range(len(results_arrays)):
60447             if results_arrays[j][i] in temp:
60548                 temp[results_arrays[j][i]] = temp[results_arrays[j][i]
606                 ] + 1
60749             else:
60850                 temp[results_arrays[j][i]] = 1
60951             results.append(max(temp, key = temp.get))
61052     return results
61153
61254
61355 def make_predicted_csv(data, csv_link = "predictions.csv"):
61456     '''takes the list data and prints its content on csv_link
61557         expects a list of either ints or female/male'''
61658     with open(csv_link, 'w', newline = '') as f:
61759         wr = csv.writer(f)
61860         for prediction in data:
61961             if type(prediction) == int:
62062                 wr.writerow(str(prediction))
62163             else:
62264                 pred = prediction.lower()
62365                 if pred == 'female':
62466                     wr.writerow(str(1))
62567                 else:
62668                     wr.writerow(str(0))
62769         f.close()
62870     return
62971
63072
63173 # In[22]:
63274
63375
63476 # testing of our algorithm
63577 # only really works in 2D
63678 '''
63779

```

```

6380 training_values = np.array([(-1, -10), (-3, -20), (-4, -40), (-1, -10)
639      , (-5, -10), (10, 100), (11, 110), (10, 90), (9, 80), (20, 150)])
64081 training_classes = np.array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1])
64182 testing_values = np.array([(-5, -5), (-2, -1), (4, 4), (9, 9)])
64283 k = 3
64384
64485
64586 testing_classes = np.array(knn(k, testing_values, training_values,
646      training_classes))
64787 print(testing_classes)
64888
64989
65090 plt.scatter(training_values[:, 0], training_values[:, 1], c =
651      training_classes, label = "training")
65291 plt.scatter(testing_values[:, 0], testing_values[:, 1], c =
653      testing_classes, marker = "v", label = "testing")
65492 plt.legend()
65593 plt.show()
65694
65795
65896 print(training_values)
65997 training_values = normalize(training_values)
66098 print(training_values)'''
66199
66200
66301 # In[13]:
66402
66503
66604 # we declare our data
66705 training_values = []
66806 training_classes = []
66907 testing_values = []
67008 testing_actual = []
67109
67210 total_lines = 1039
67311
67412 number_of_training = 780
67513
67614
67715 link = r"C:\Users\vikin\Documents\STS\Statistisk maskininl rning\
678      projects\1\train.csv"
67916 short = r"C:\Users\vikin\Documents\STS\Statistisk maskininl rning\
680      projects\1\short.csv"
68117
68218 # we fill our data
68319 file = open(link)
68420 csv_reader = csv.reader(file)
68521 next(csv_reader) # skip the header
68622
68723 i = 0
68824 for row in csv_reader:
68925     if i < number_of_training:
69026         training_values.append(tuple(row[0 : -1]))
69127         training_classes.append(row[-1])
69228     else:
69329         testing_values.append(tuple(row[0 : -1]))
69430         testing_actual.append(row[-1])
69531     i += 1
69632
69733 # normalization of the data
69834 training_values = np.array(training_values).astype(np.float)
69935 training_values = normalize(training_values)
70036
70137 testing_values = np.array(testing_values).astype(np.float)
70238 testing_values = normalize(testing_values)

```



```

70339
70440
70541 # # Finding the Best K Value
70642 # We'll test K values in [1, 41] and pick a K value based on accuracy.
70743
70844 # In[29]:
70945
71046
71147 for k in range(1, 42):
71248     print("K:", k)
71349
71450     testing_classes = np.array(knn(k, testing_values, training_values,
715     training_classes))
71651     print(results(testing_classes, testing_actual))
71752
71853
71954 # # About the Data
72055 # To test only certian attributes:
72156 #     modify the attributes list to contain the indices you want to
722     test and move it like below
72357 #
72458 #     altered_testing = testing_values[:, attributes]
72559 #     altered_training = training_values[:, attributes]
72660 #
72761 #
72862 # ## Key:
72963 # * Number words female: 0,
73064 # * Total words: 1,
73165 # * Number of words lead: 2,
73266 # * Difference in words lead and co-lead: 3,
73367 # * Number of male actors: 4,
73468 # * Year: 5,
73569 # * Number of female actors: 6,
73670 # * Number words male: 7,
73771 # * Gross: 8,
73872 # * Mean Age Male: 9,
73973 # * Mean Age Female: 10,
74074 # * Age Lead: 11,
74175 # * Age Co-Lead: 12
74276
74377 # In[61]:
74478
74579
74680 # let's run this shit
74781 #for i in range(13):
74882 #print(i)
74983 attributes = [0,7] #[0, 5, 7, 8]
75084
75185 k = 3
75286 altered_testing = testing_values[:, attributes]
75387 altered_training = training_values[:, attributes]
75488
75589 testing_classes = np.array(knn(k, altered_testing, altered_training,
756     training_classes))
75790 print(results(testing_classes, testing_actual))
75891
75992
76093 # # Sampling Without Replacement
76194 #
76295 # We split our matrix into k subsets and predict based on that. We the
763     choose the most common prediction.
76496
76597 # In[35]:
76698
76799

```

```

76800 k_split = 10
76901 k_neighbors = 3
77002
77103 #print(len(training_values))
77204 #print(training_values)
77305
77406 #print(len(training_classes))
77507 #print(training_classes)
77608
77709 split_training_values = split_into_k(training_values, k_split)
77810 #print(split_training_values)
77911 split_training_classes = split_into_k(training_classes, k_split)
78012 #print(split_training_classes)
78113
78214 sampling_results = [0] * k_split
78315
78416 #print(split_training_values[i])
78517 #print(split_training_classes[i])
78618
78719 for i in range(k_split):
78820     sampling_results[i] = knn(k_neighbors, altered_testing,
789     split_training_values[i], split_training_classes[i])
79021
79122
79223 #print(cross_validation_results)
79324 sampling_predictions = combine_results(sampling_results)
79425
79526
79627 print(results(sampling_predictions, testing_actual))
79728
79829
79930 # # Making a CSV File
80031 #
80132 #
80233
80334 # In[71]:
80435
80536
80637 import os
80738 print(os.getcwd())
80839
80940 d = [1,0,1,0,1,0,1,0,2,3,1,1]
81041 #d = ["male", "female", "MAle", "Female"]
81142
81243
81344 make_predicted_csv(d)
81445
81546
81647 # In[64]:
81748
81849
81950 # 2D plot
82051
82152
82253
82354 blue_class = np.array([(1,1),(4,10),(5,6),(10, 3),(2,4)])
82455 red_class = np.array([(5,1),(6,4),(2,6),(3, 10),(6.5,5)])
82556 yellow_class = np.array([(5, 5)])
82657
82758
82859
82960 fig, ax = plt.subplots()
83061 #plt.xticks([])
83162 #plt.yticks([])
83263

```

```

8334 ax.set(xlim=(0, 11), ylim = (0, 11))
8345 ax.set_aspect('equal')
8356 ax.axes.xaxis.set_visible(False)
8367 ax.axes.yaxis.set_visible(False)
8378
8389 ax.scatter(blue_class[:, 0], blue_class[:, 1], c = "b", marker = "v",
839 label = "blue class")
8400 ax.scatter(red_class[:, 0], red_class[:, 1], c = "r", marker = "v",
841 label = "red class")
8421 ax.scatter(yellow_class[:, 0], yellow_class[:, 1], c = "y", label = "
843 unlabeled")
8442 #ax.scatter([x^2 + y^2 = 1])
8453 cir1 = plt.Circle((5, 5), 1, color='b',fill=False)
8464 #ax.add_artist(cir1)
8475 cir2 = plt.Circle((5, 5), 1.5, color='r',fill=False)
8486 ax.add_artist(cir2)
8497 plt.title('Classification with k = 3')
8508
8519
8520
8531 ax.legend()
8542
8553 fig.savefig('plot2.png')
8564
8575 #ax.show()
8586
8597 '''
8608
8619 training_values = np.array([(-1, -10), (-3, -20), (-4, -40), (-1, -10)
862 , (-5, -10), (10, 100), (11, 110), (10, 90), (9, 80), (20, 150)])
8630 training_classes = np.array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1])
8641 testing_values = np.array([(-5, -5), (-2, -1), (4, 4), (9, 9)])
8652 k = 3
8663
8674
8685 testing_classes = np.array(knn(k, testing_values, training_values,
869 training_classes))
8706 print(testing_classes)
8717
8728
8739 plt.scatter(training_values[:, 0], training_values[:, 1], c =
874 training_classes, label = "training")
8750 plt.scatter(testing_values[:, 0], testing_values[:, 1], c =
876 testing_classes, marker = "v", label = "testing")
87701 plt.legend()
8782 plt.show()
8793
8804
8815 print(training_values)
8826 training_values = normalize(training_values)
8837 print(training_values)'''

884 1 #!/usr/bin/env python
885 2 # coding: utf-8
886 3
887 4 # In [ ]:
888 5
889 6
890 7 import pandas as pd
891 8 import numpy as np
892 9 import matplotlib.pyplot as plt
893 10 import csv
894 11
895 12 #scikit
896 13 import sklearn.preprocessing as skl_pre

```

```

89714 import sklearn.linear_model as skl_lm
89815 import sklearn.model_selection as skl_ms
89916
90017 from IPython.core.pylabtools import figsize
90118 figsize(10, 6)
90219
90320
90421 # In[ ]:
90522
90623
90724 # Read data
90825 directory = 'directory of dataset'
90926 traindata = pd.read_csv(directory, na_values='?', dtype={'Lead': str})
910      .dropna().reset_index()
91127 #traindata.info()
91228
91329 # Preprocessing data
91430 np.random.seed(1)
91531
91632 # Splitting up dataset in training and test data: 538 training, 500
917      test
91833 trainIndex = np.random.choice(traindata.shape[0], size=538, replace=
919      False)
92034
92135 trainIndexBool = traindata.index.isin(trainIndex) # generates a np
922      array with True/False if corresponding index in in trainIndex
92336 train = traindata.iloc[trainIndexBool] # training set of 538
92437 test = traindata.iloc[~trainIndexBool] # test set of 500 (1038-538)
92538
92639 # Feature selection
92740 features = ['Number words female', 'Total words', 'Number of words
928      lead', 'Difference in words lead and co-lead', 'Number of male
929      actors', 'Year', 'Number of female actors', 'Number words male',
930      'Gross', 'Mean Age Male', 'Mean Age Female', 'Age Lead', 'Age Co-
931      Lead']
93241
93342 # Training set
93443 X_train = train[features] # input
93544 Y_train = train['Lead'] # output
93645
93746 # Test set
93847 X_test = test[features]
93948 Y_test = test['Lead']
94049
94150
94251 # In[ ]:
94352
94453
94554 # Learn model
94655 log_reg_model_ridge = skl_lm.LogisticRegression(solver='liblinear',
947      penalty='l2', C=1000) # Ridge Regression
94856 log_reg_model_ridge.fit(X_train,Y_train)
94957 #print('Model coefficients, Ridge Logistic Regression: \n',
950      log_reg_model_ridge.coef_, '\n')
95158
95259 # Prediction
95360 r = 0.5
95461 prediction = np.where(prob[:,0] >= r, 'Female','Male')
95562
95663 # Confusion matrix
95764 print('Confusion matrix:\n', pd.crosstab(prediction, Y_test), '\n')
95865
95966 # Accuracy
96067 print(f'Accuracy: {np.mean(prediction == Y_test):.3f}', '\n') # rounds
961      up to 3 decimals

```

```

96258
96359
96470 # In[ ]:
96571
96672
96773 # Compare Logistic regression models with/without regularization
96874 log_reg_model_lasso = skl_lm.LogisticRegression(solver='liblinear',
969     penalty='l1', C=1000) # Lasso Regression
97075 log_reg_model_ridge = skl_lm.LogisticRegression(solver='liblinear',
971     penalty='l2', C=1000) # Ridge Regression
97276 log_reg_model_none = skl_lm.LogisticRegression(solver='lbfgs', penalty
973     ='none', max_iter=2500) # no regularization
97477
97578 models = []
97679 models.append(log_reg_model_lasso)
97780 models.append(log_reg_model_ridge)
97881 models.append(log_reg_model_none)
97982
98083 # learn model from training data
98184 for model in models:
98285     model.fit(X_train,Y_train)
98386
98487 print('Model coefficients, Lasso Logistic Regression: \n',
985     log_reg_model_lasso.coef_, '\n')
98688 print('Model coefficients, Ridge Logistic Regression: \n',
987     log_reg_model_ridge.coef_, '\n')
98889 print('Model coefficients, Logistic Regression: \n',
989     log_reg_model_ridge.coef_, '\n')
99090
99191
99292 # In[ ]:
99393
99494
99595 pred_prob_lasso = log_reg_model_lasso.predict_proba(X_test)
99696 pred_prob_ridge = log_reg_model_ridge.predict_proba(X_test)
99797 pred_prob_none = log_reg_model_none.predict_proba(X_test)
99898
99999 models_prob = {}
100000 models_prob[log_reg_model_lasso] = pred_prob_lasso
100101 models_prob[log_reg_model_ridge] = pred_prob_ridge
100202 models_prob[log_reg_model_none] = pred_prob_none
100303
100404 # Descision
100505 r = 0.5
100606
100707 for model, prob in models_prob.items():
100808     print('Class order in the model: ', model.classes_)
100909
101010     # Print first 5 predictions
101111     print('Examples of predicted probabilities for the above
1012     classes: \n', item[1][0:5], '\n')
101312     # print(f'Average probability for Female lead is {100*(sum(
1014     prob[:,0])/len(prob)):.0f} %')
101513     # print(f'Average probability for Male lead is {100*(sum(prob
1016    [:,1])/len(prob)):.0f} %', '\n')
101714
101815     # Prediction
101916     prediction = np.where(prob[:,0] >= r, 'Female','Male')
102017
102118     # Confusion matrix
102219     print('Confusion matrix:\n', pd.crosstab(prediction, Y_test),
1023     '\n')
102420
102521     # Accuracy

```

```

102622         print(f'Accuracy: {np.mean(prediction == Y_test):.3f}', '\n')
1027         # rounds up to 3 decimals
102823         # print('Accuracy: ', np.mean(prediction == Y_test))
102924
103025         # Precision
103126
103227         # Recall
103328
103429         # F1
103530
103631
103732 # In[ ]:
103833
103934
104035 # decide positive resp negative class
104136 pos_class = 'Male'
104237 neg_class = 'Female'
104338
104439 # count num of positive resp negative samples in test
104540 P = np.sum(Y_test == pos_class) # num of positive samples in test
104641 N = np.sum(Y_test == neg_class) # num of negative samples in test
104742
104843 # find indices with male/positive class in data
104944 pos_class_index = np.argwhere(log_reg_model.classes_ == pos_class).
1050     squeeze()
105145
105246 # lists to append TP / FN rate for different r
105347 true_pos_rate = []
105448 false_pos_rate = []
105549
105650 # r [0,1], dvs r= 0.00, 0.01,...,0.1,0.11,...,0.99,1.0
105751 threshold = np.linspace(0.00, 1, 101) # start=0, stop=1, 101 num
1058     evenly spaced samples
105952
106053 for model, prob in models_prob.items():
106154     for r in threshold:
106255         prediction = np.where(prob[:,pos_class_index] > r, pos_class,
1063         neg_class)
106456         TP = np.sum((prediction == pos_class) & (Y_test == pos_class))
1065         #True Positive
106657         true_pos_rate.append(TP/P)
106758         FP = np.sum((prediction == pos_class)&(Y_test == neg_class)) #
1068         False Positive
106959         false_pos_rate.append(FP/N)
107060
107161         plt.plot(false_pos_rate, true_pos_rate);
107262         for i in [0,1,10,50,98,100]:
107363             plt.text(false_pos_rate[i], true_pos_rate[i], f'r={threshold[i]
1074             }:.2f}')
107564         plt.xlim([0, 1])
107665         plt.ylim([0, 1.1])
107766         plt.xlabel('False positive rate')
107867         plt.ylabel('True positive rate');
107968
108069 # print('false_pos_rate',false_pos_rate)
108170 # print('true_pos_rate', true_pos_rate)
108271
108372
108473 # In[ ]:
108574
108675
108776 # Cross validation
108877
108978 n_fold = 10
109079

```

```

109180 models = []
109281 models.append(skl_lm.LogisticRegression(solver='liblinear', penalty='
1093     l1', C=1000)) # Lasso Regression
109482 models.append(skl_lm.LogisticRegression(solver='liblinear', penalty='
1095     l2', C=1000)) # Ridge Regression
109683 models.append(skl_lm.LogisticRegression(solver='lbfgs', penalty='none',
1097     , max_iter=2500)) # no regularization
109884
109985 missclassification = np.zeros((n_fold, len(models)))
110086 cv = skl_ms.KFold(n_splits=n_fold, random_state = 1, shuffle = True)
110187
110288 for i, (train_index, val_index) in enumerate (cv.split(X_train)):
110389     X_trainfold, X_val = X_train.iloc[train_index], X_train.iloc[
1104     val_index]
110590     Y_trainfold, Y_val = Y_train.iloc[train_index], Y_train.iloc[
1106     val_index]
110791
110892     for m in range(np.shape(models)[0]): #try different models
110993         model = models[m]
111094         model.fit(X_trainfold, Y_trainfold)
111195         prediction = model.predict(X_val)
111296         missclassification[i,m] = np.mean(prediction != Y_val)
111397
111498 print(missclassification)
111599 plt.boxplot(missclassification)
111600 plt.title('cross validation error for different methods')
111701 plt.xticks(np.arange(4)+1, ('Logistic with Lasso', 'Logistic with
1118     Ridge', 'Logistic no regularization'))
111992 plt.ylabel('validation error')
112093 plt.show()
112104
112295
112396 # In[ ]:
112497
112598
112699 # Cross validation, changing pentalty parameter C (or lambda)
112700
112801 n_fold = 10
112902 C = [1,10,50,100,400,999,1000,2000,5000]
113003
113104 missclassification = np.zeros((n_fold, len(C)))
113205 cv = skl_ms.KFold(n_splits=n_fold, random_state = 1, shuffle = True)
113306
113407 for i, (train_index, val_index) in enumerate (cv.split(X_train)):
113508     X_trainfold, X_val = X_train.iloc[train_index], X_train.iloc[
1136     val_index]
113709     Y_trainfold, Y_val = Y_train.iloc[train_index], Y_train.iloc[
1138     val_index]
113910
114011     for j in range(len(C)): #try different models/C
114112         model = skl_lm.LogisticRegression(solver='liblinear', penalty=
1142     'l2', C=C[j]) # Ridge Regression
114313         model.fit(X_trainfold, Y_trainfold)
114414         prediction = model.predict(X_val)
114515         missclassification[i,j] = np.mean(prediction != Y_val)
114616
114717 #print(missclassification, '\n')
114818 plt.boxplot(missclassification)
114919 plt.title('cross validation error for different methods')
115020 plt.xticks(np.arange(9)+1, ('c=1', 'c=10', 'c=50', 'c=100', 'c=400', '
1151     c=999', 'c=1000', 'c=1001', 'c=5000'))
115221 plt.ylabel('validation error')
115322 plt.show()

```