# Machine Learning in Datatscience - Project Report
# Project 1

Matthias Krug
Alper Savas
Ali Bektas

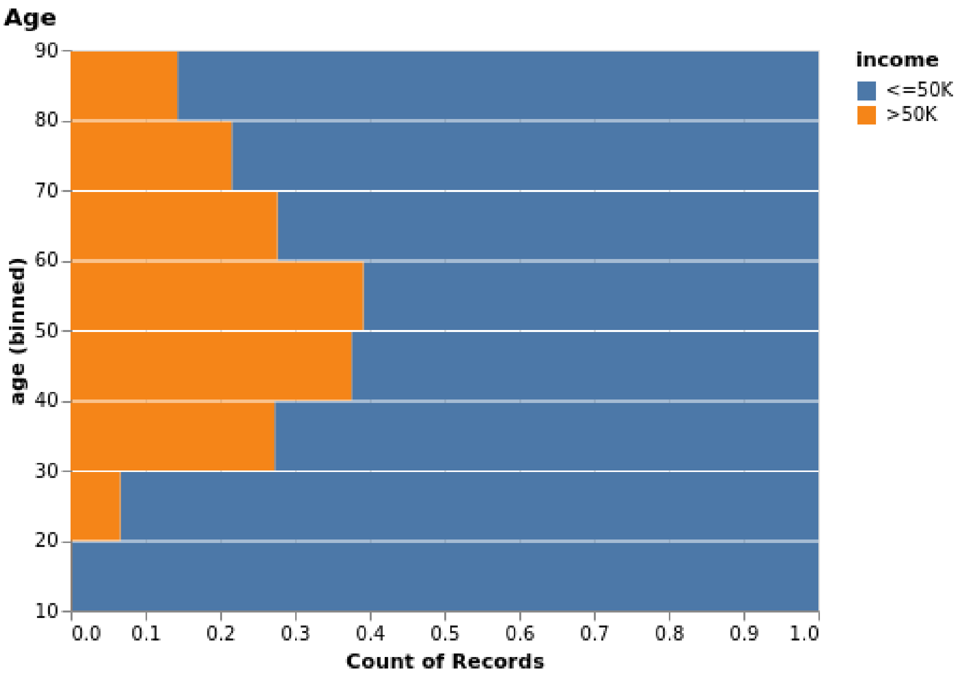
December 21, 2021

## Exploratory data analysis

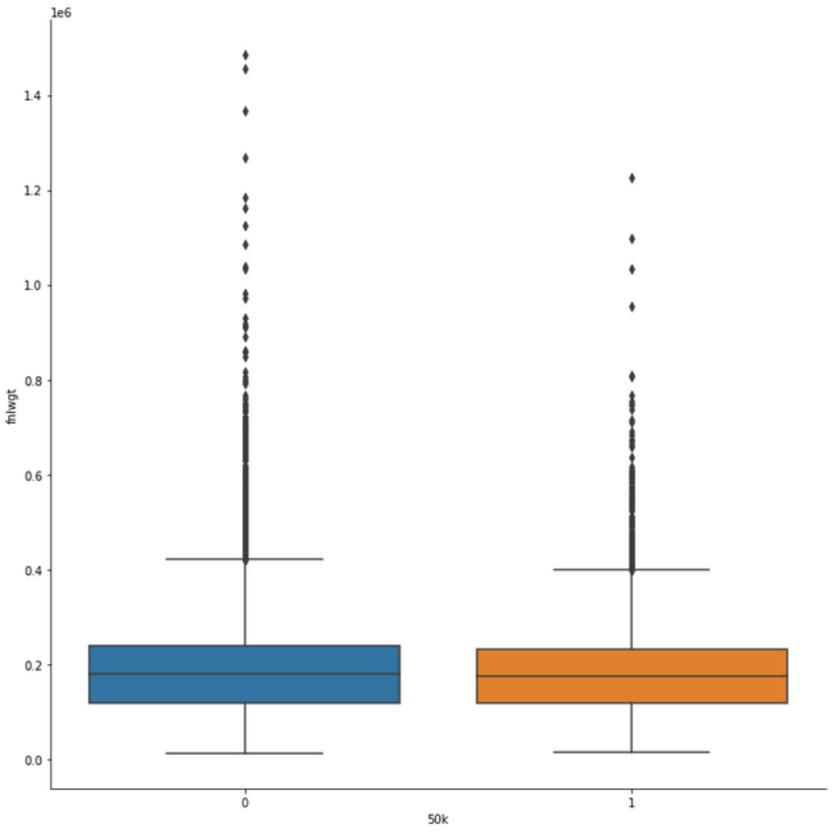
### Univariate analysis

#### Plot the value of each feature (boxplot, histogram, etc.). What do you observe? Check for class imbalance. If existing, what are its effects? How can you cope with it?

Lets go through all the Features one by one and note our findings. We will color the people making over 50k orange and the rest Blue. Most Graphs will be normalized bargraphs. For this first part the Code we used to analyse and visualize the Data is in the File "Data.ipynb".

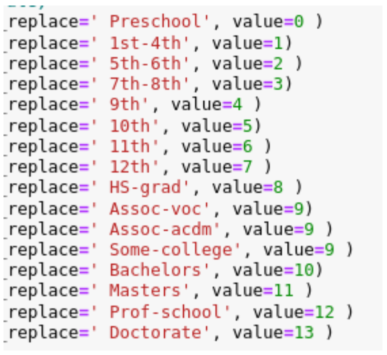


So we see that there is a correlation between income and age. With people in Around their 50s making most money. Unsurprisingly as those people are at the end of their careers and had enough time to already get many raises.



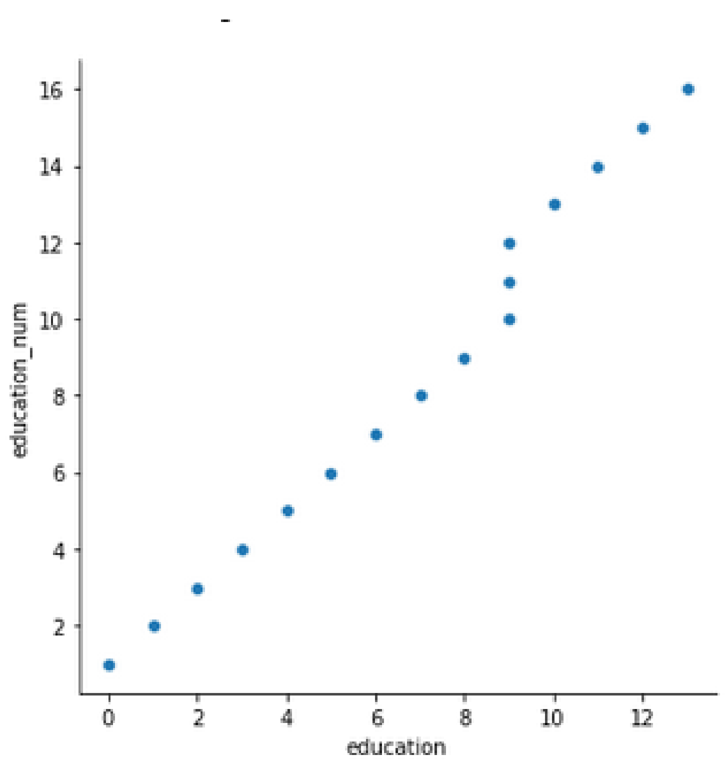
As we see in this Boxplot whatever "fnlwgt" is, it is not correlated with whether people make or not make more than 50k of income. Both Boxplots are basically identical therefore the distribiution of "fnlwgt" is equal among both classes. Therefore we will ignore that attribiute from now on and delete it from our data.

Education is obviously important but also obviously nominal data. But its also Ordinal as someone with a PHD spend more time in school than someone who competed just preschool. Lets inspect this by first turning the ordinal attribute Education into a numerical Attribute using this assignment roughly ordered by years of education:

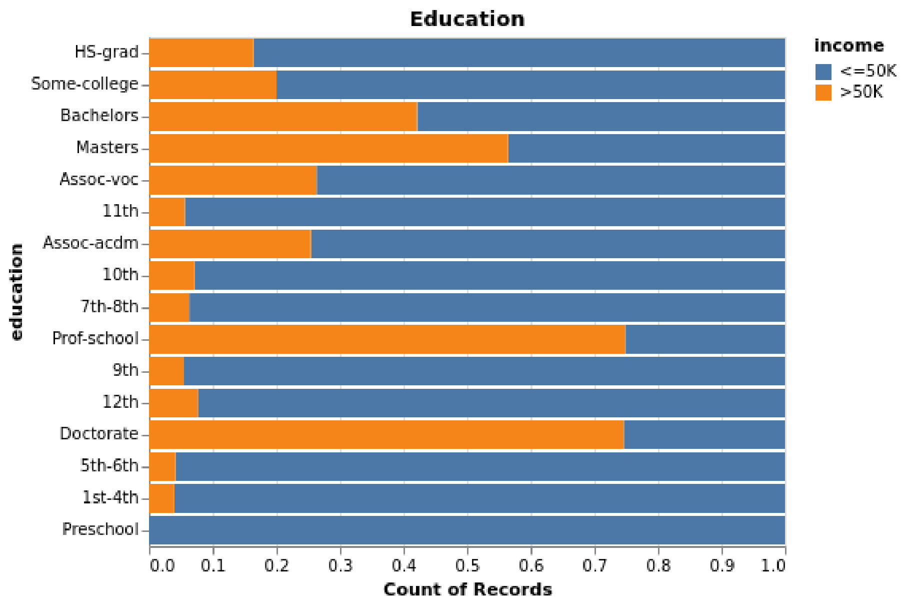


```
_replace=' Preschool', value=0 )
_replace=' 1st-4th', value=1)
_replace=' 5th-6th', value=2 )
_replace=' 7th-8th', value=3)
_replace=' 9th', value=4 )
_replace=' 10th', value=5)
_replace=' 11th', value=6 )
_replace=' 12th', value=7 )
_replace=' HS-grad', value=8 )
_replace=' Assoc-voc', value=9)
_replace=' Assoc-acdm', value=9 )
_replace=' Some-college', value=9 )
_replace=' Bachelors', value=10)
_replace=' Masters', value=11 )
_replace=' Prof-school', value=12 )
_replace=' Doctorate', value=13 )
```

We have the attribute education_num which appears to be a numerical attribute describing education. Lets compare our artificial numerical education with the given numerical education attribute:

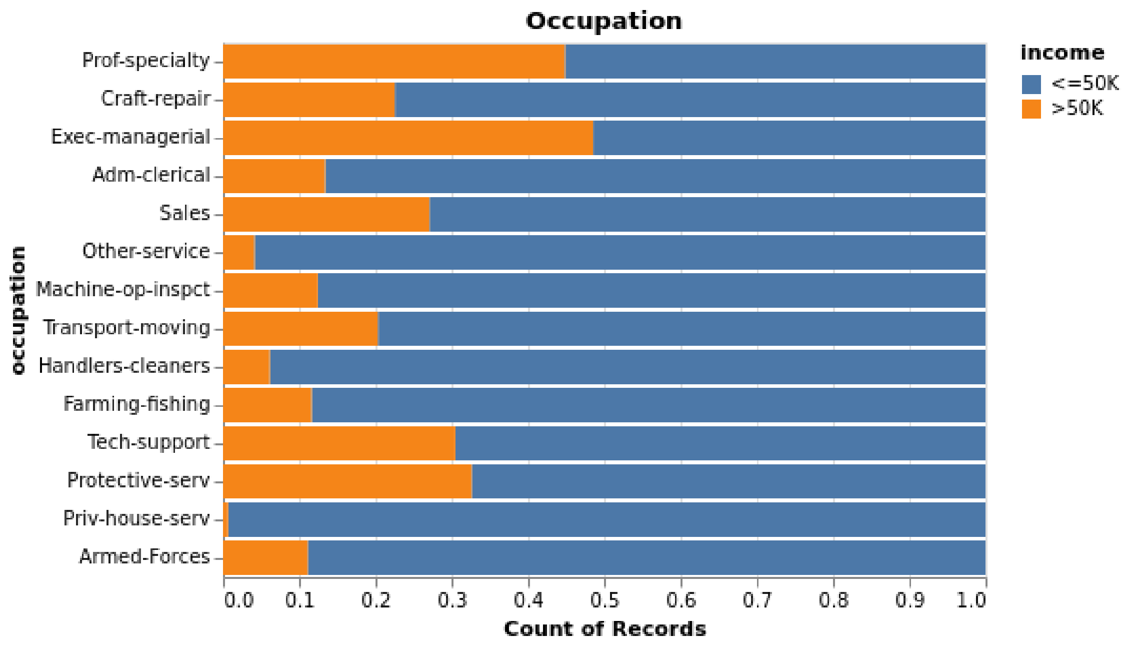


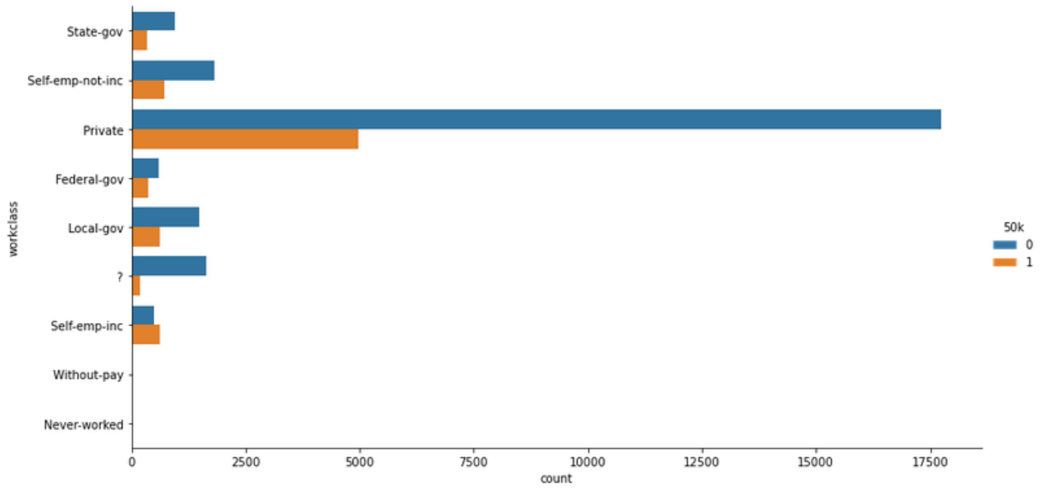
They are basically the exact same. So instead of dealing with redundant nominal data or with our replacement we will just delete education and use the given education_num instead as it encodes the exact same information, and less dimensions are always great.



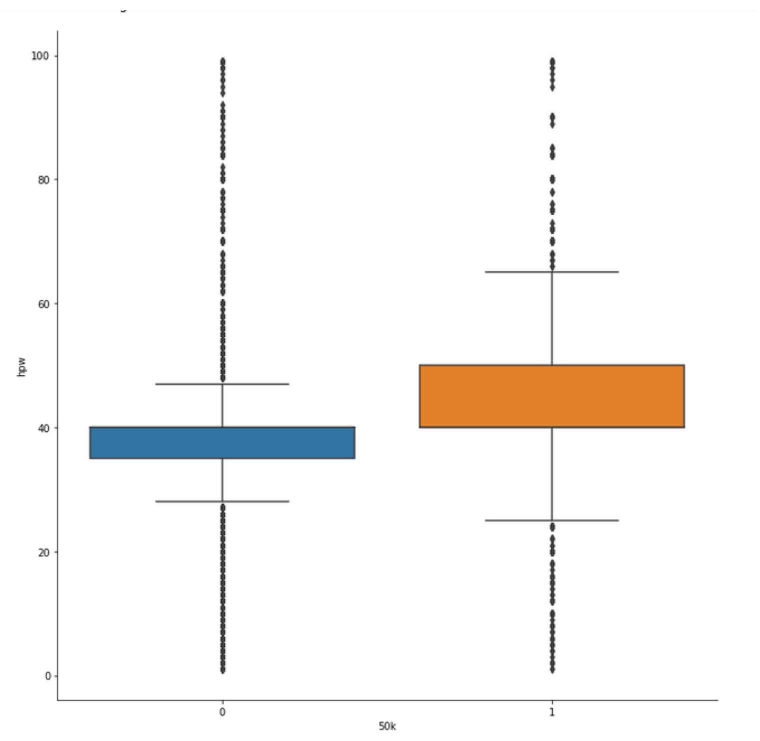
And as we see education is an extremely important attribute that tells us a lot about whether someone will make more than 50k. People with Master degrees, Professional Schools (like Law or Med School) or PHDs have a far far higher chance of making >50k.

Lets now also look at Occupation. We see that there are a few high paying occupations, especially Exec-Managerial and Prof-Speciality with over 40% over 50k are noteworthy, and a few very low paying occupations below 10% like Handlers-Cleaners, Private-House-Service and Other-Service. We will make our lives simpler by turning that into a simple numerical variable assigning +1 for the 2 high paying fields, -1 for the 3 low paying fields and 0 for all other fields which are in the middle. We also could have used a more sophisticated method by assigning each field the appropriate percentage between 0-1 in accordance with the percentage of people making over 50k.

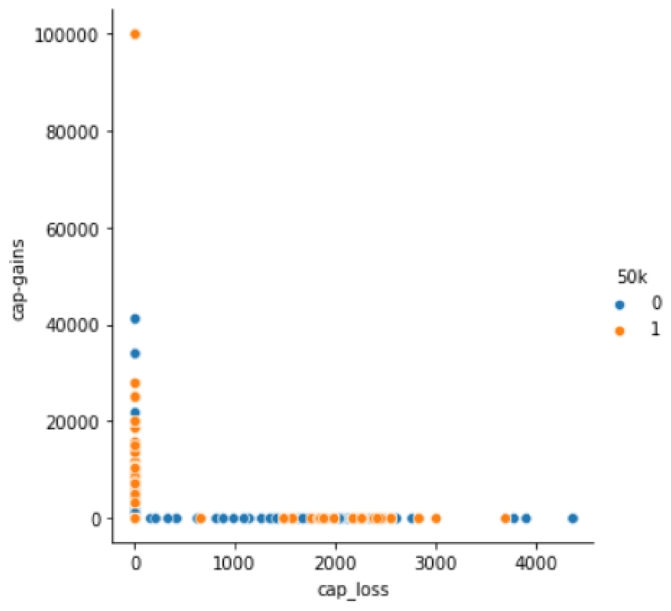




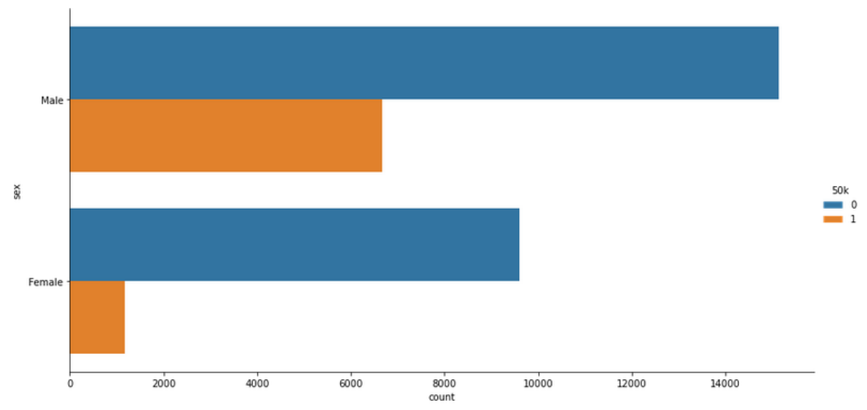
This Barplot is not normalized to show us that there is a strong class imbalance We see that workforce is strongly imbalanced towards regularly employed people. The other 2 groups that are notable are government employees and self employed people. There are some groups that make more than others. A simple way to aggregate the Data is to create 3 binary variables: IsPrivate, IsGoverment and IsSelfEmpoyed to cover the important groups.



So this Boxplot shows us that the hours someone works per week do correlate with whether someone makes over 50k or not. This kind of makes sense for 2 people who make roughly the same per hour, the one working more hours will make more and might therefore just pass the 50k mark at the end of the year while the other stays below 50k.



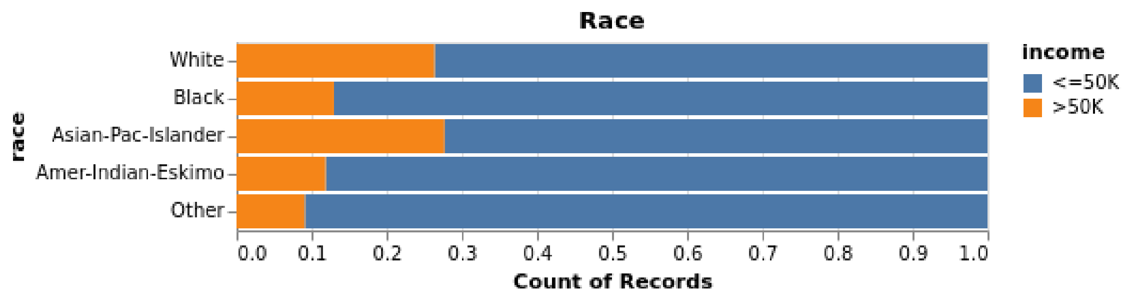
This Scatterplot shows us 2 things: **A:** Capital gains and capital losses are aggregate. You either have one or the other. so if a person makes 5k$ on a Stock and then looses 10k$ on a Stock then this will count as 5k in capital losses. This means we can just subtract capital losses from capital gains and get 1 single aggregate number without loss of information. **B:** People making capital gains, or high capital losses seem to make more than people making low ones.



As we see there is a big class imbalance between Man and Woman in the data, both for all people and for people making over 50k males are disproportionally over represented. Here are the Numbers:

| People | People >50k | Woman | Man | Woman >50k | Man >50k |
|---|---|---|---|---|---|
| 32561 | 7841 | 10771 | 21790 | 1179 | 6662 |

So there are 2x more man than woman in the data and 6x more man making over 50k than woman. This has consequences we will have to address in 3.5. Also only 1/4 of all people makes over 50k so an "dumb" classifier which just predicts ≤50k for every instance would have a 75% accuracy.



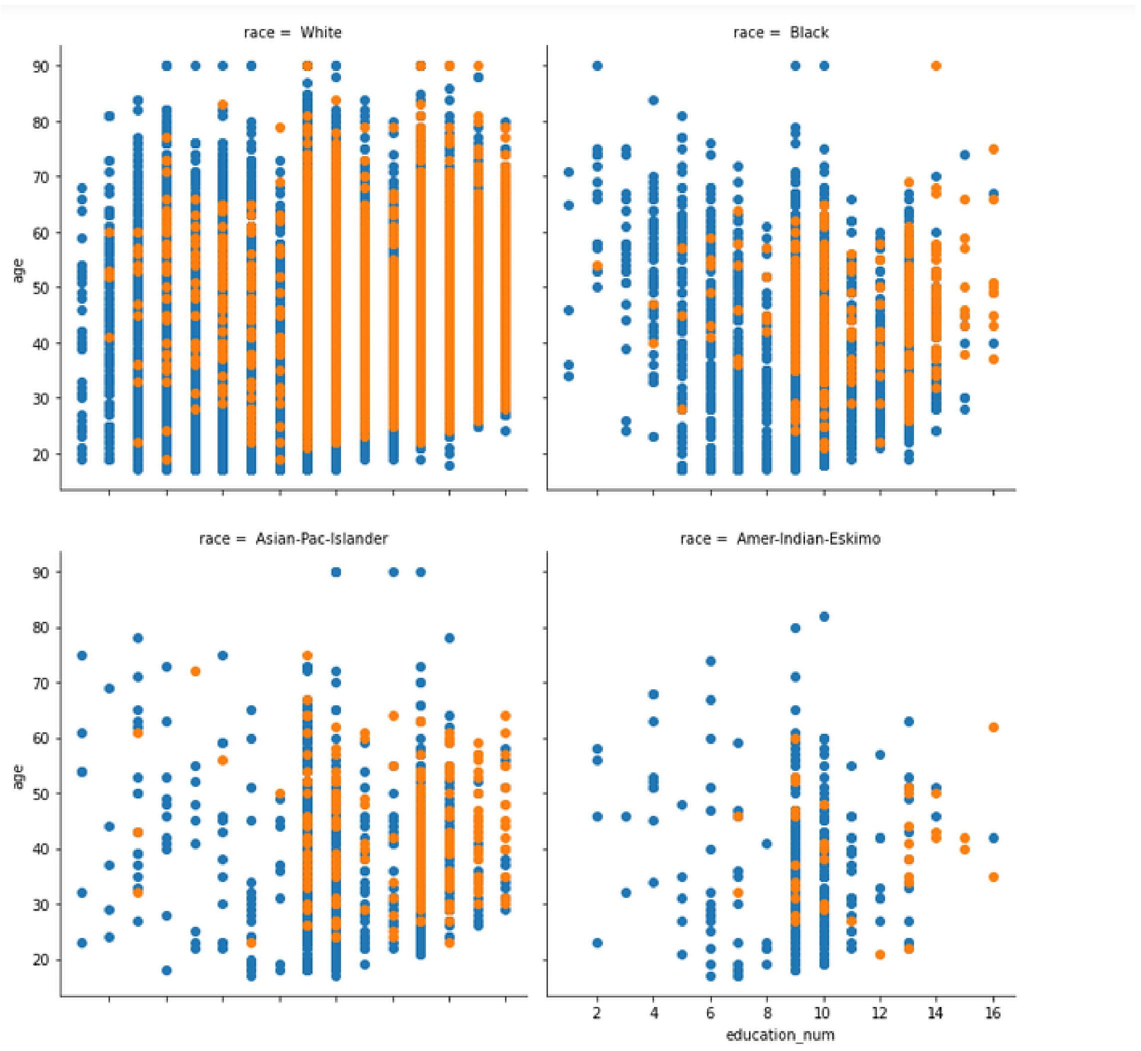
Here we see a strong Raceimbalance. With White and Asian-Pacific-Islander having roughly 30% of their groups over 50k, while the other categories being behind. If we look at this in form of scatterplots across both age and education (We left out "Other" to focus on the main Races) we see a few things:

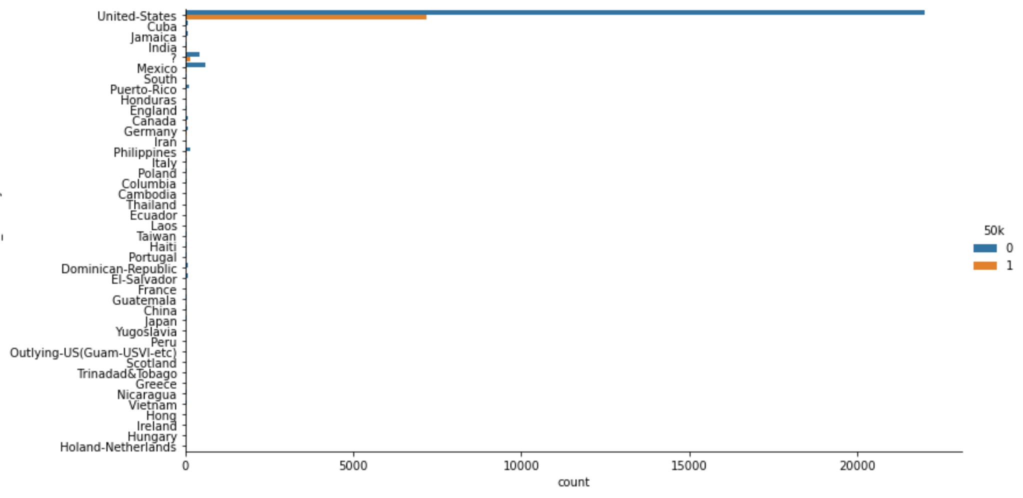
**A:** Whites are the Majority across all ages and education levels. As Expected Income over 50k correlates strongly with education and clusters around ages 50-60.

**B:** Blacks with higher education are strongly underrepresented. Those that have high education though seem to also have Incomes over 50k in general.

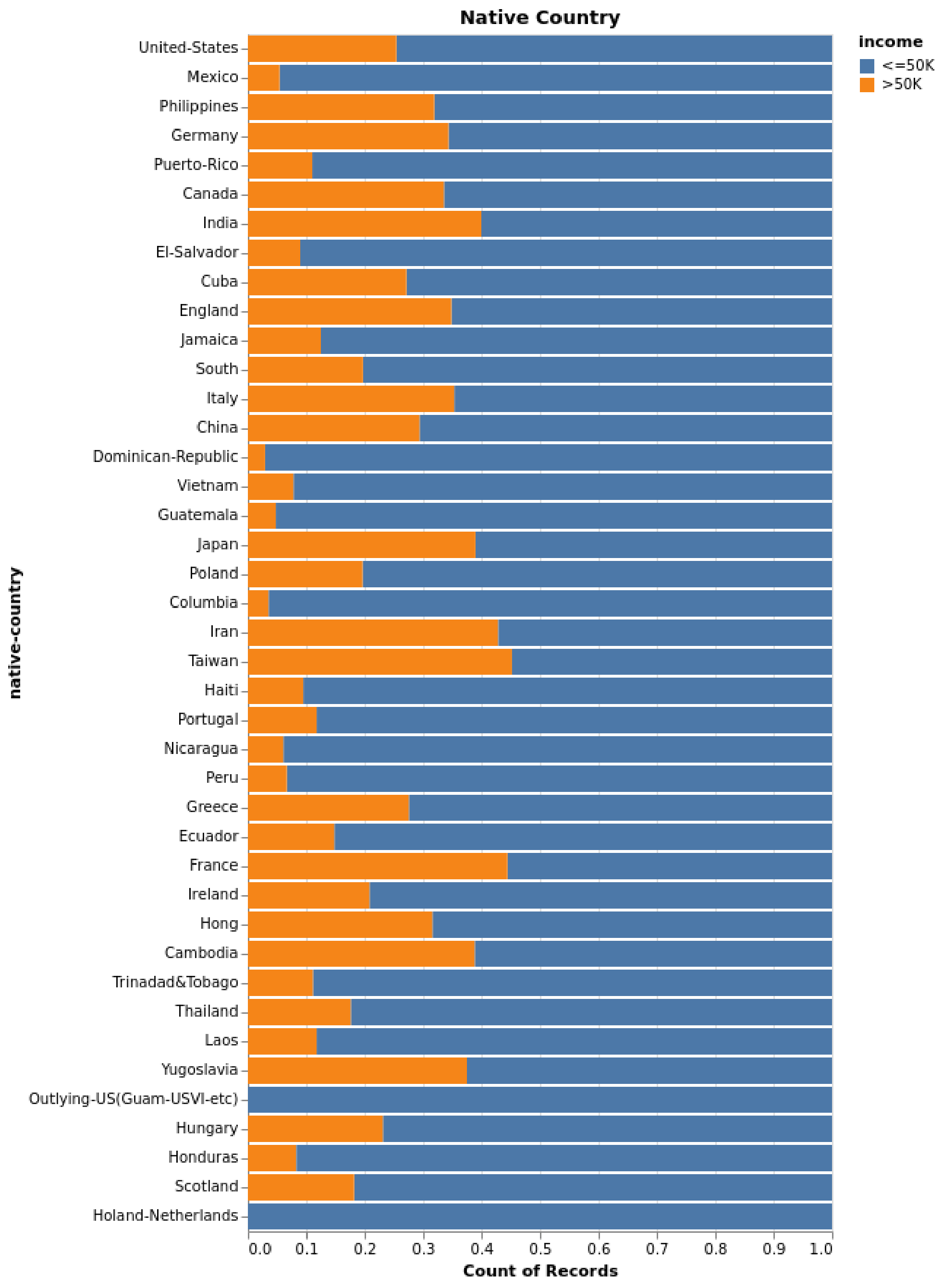
**C:** Asian-Pacific-Islanders are strongly over represented in higher education which as we know strongly correlates with high income.

**D:** American-Indian-Eskimos are underrepresented in general but also cluster in Mid level education which stops them from reaching high Income levels.

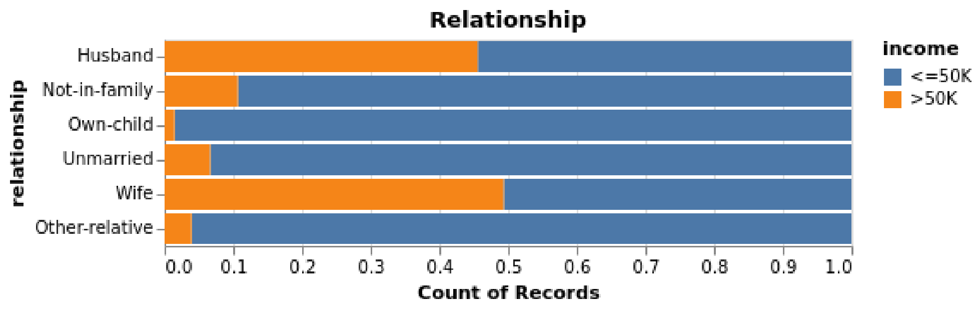




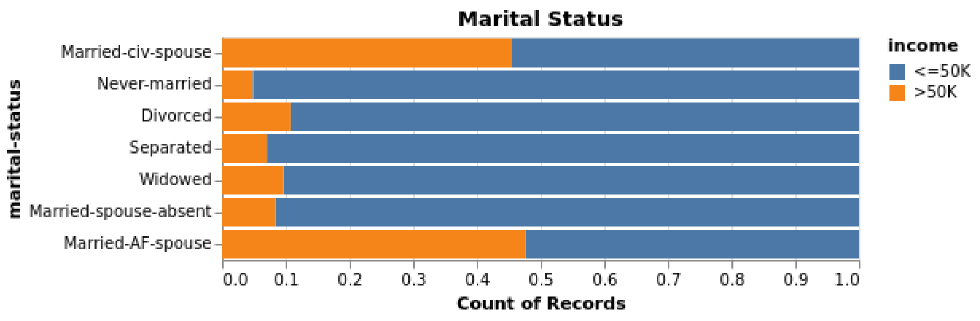
This is an extreme class imbalance. So we will have to normalize. Once we do we see that there are countries of origin like Taiwan, France, Japan, Iran etc with many people with incomes over 50k and other countries like Haiti, Columbia or Mexico with few people making over 50k. But all in all we will delete Country as an attribute as turning all that Nominal Data into Numerical Data would give us many additional Dimensions which we want to avoid, and as we only have very few examples for countries which are not USA or Mexico the statistical significance of the Data can not be guaranteed.



If we look at Relation Status we see that Married people, both wife's and husbands are far more likely to make over 50k than not married people:

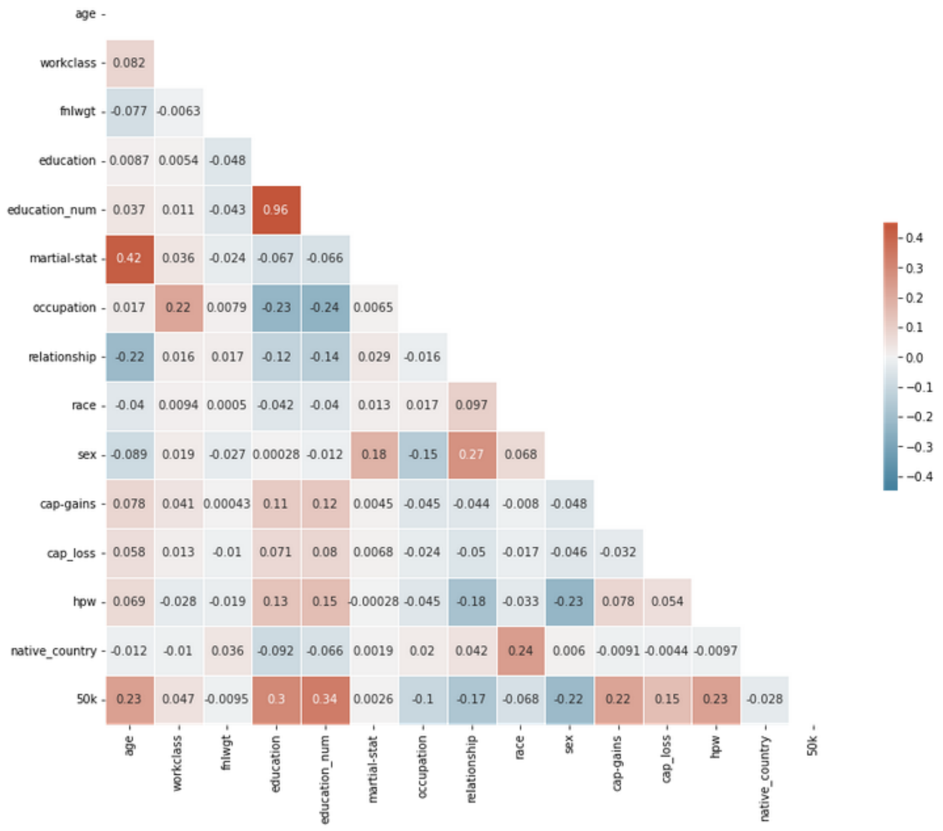


Martial Status confirms this. Therefore we will combine those 2 attributes into 1 simple binary attribute: **IsMarried**. This allows us to save many dimensions and still carry most of the relevant Information.



## Bivariate analyis

And now for the correlation matrix:



Nothing really surprising here. Education and Education_Num are the same, making over 50k per year correlates with the things we have found out that it correlates with, fnlwgt correlates with nothing really, the only 2 notable things we haven not discussed yet is that age correlates with Martial Status, which is kind of obvious in hindsight and that Age correlates with marital and relationship status which is a bit weird as one would expect Man and Woman being equally likely married or in relationships / single. Finally Race and native Country being closely correlated is kind of obvious too.

## Feature selection - Dataset preparation

We already described in the data-analysis part how we wanted to select our features and prepare our data, so for details scroll up. **But:**

*Comment with Hindsight:* We tried the classifiers with data where we combined the variables as described in our analysis and with simple One Hot Encoding. Using One Hot Encoding we did see a bit better performance. Apparently the sklearn library implementations of the classifiers are optimized for people to not think to much and just dump data in.

## Classification

### Evaluation measures

*Comment on the performance of the different learners, also with regard to class (im)balance.*

All Code for the Clasifiers can be found in: "Classificationsmodified.ipynb".

Before doing parameter tuning in a systematic way we evaluated the different classier with the some initial hyper parameter values to get a some first impression about the performance of the different classifiers on the adult income dataset.

We trained the classifiers with the following hyper parameter values:

```
dt = tree.DecisionTreeClassifier(criterion='gini', max_depth=10)
knn = KNeighborsClassifier(n_neighbors = 10)
nb = GaussianNB()
perc = Perceptron(tol=1e-3, random_state=0)
```

Because of performance reasons we trained the SVM classifier with a subset of only 1000 datapoints and same proportion of classes labels about 30/70 (true/negative) class.

Figure 1 to 5 is showing the performance metrices for the different classifier.

With this initial setup the Decision Tree (DT) classifier is showing the best performance overall in terms of accuracy and also with regards to the class imbalance.

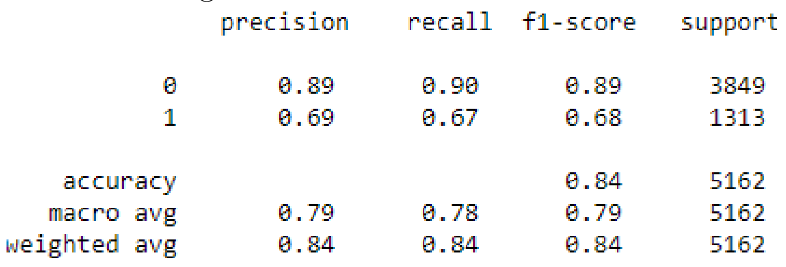


Figure 1: Decision Tree evaluation

```
              precision    recall  f1-score   support

           0       0.89      0.90      0.89      3849
           1       0.69      0.67      0.68      1313

    accuracy                           0.84      5162
   macro avg       0.79      0.78      0.79      5162
weighted avg       0.84      0.84      0.84      5162
```

One reason why the Decision Tree classifier is performing better then the other classifiers could be that DTs tend to have a high bias in general. With a tree depth of 10 it could be that the classifier is making splits where one side is containing only a few datapoints. Another reason may be the inherent ability of Decision Trees to handle categorical data well, something that other classifiers struggle with.

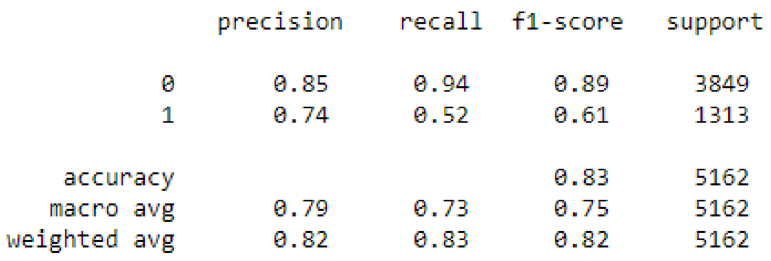


Figure 2: KNN evaluation

```
              precision    recall  f1-score   support

           0       0.85      0.94      0.89      3849
           1       0.74      0.52      0.61      1313

    accuracy                           0.83      5162
   macro avg       0.79      0.73      0.75      5162
weighted avg       0.82      0.83      0.82      5162
```

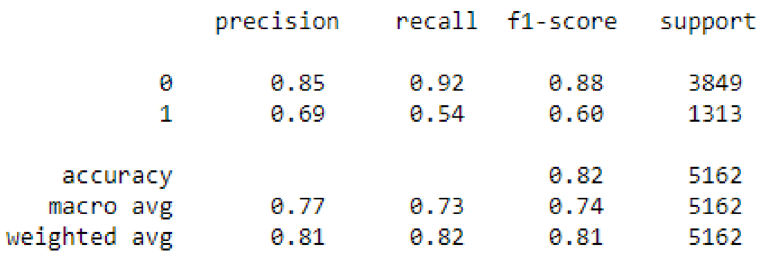


Figure 3: Naive Bayes evaluation

```
              precision    recall  f1-score   support

           0       0.85      0.92      0.88      3849
           1       0.69      0.54      0.60      1313

    accuracy                           0.82      5162
   macro avg       0.77      0.73      0.74      5162
weighted avg       0.81      0.82      0.81      5162
```

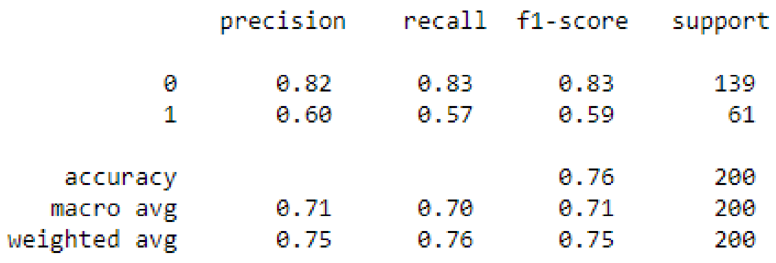


Figure 4: SVM evaluation

```
              precision    recall  f1-score   support

           0       0.82      0.83      0.83       139
           1       0.60      0.57      0.59        61

    accuracy                           0.76       200
   macro avg       0.71      0.70      0.71       200
weighted avg       0.75      0.76      0.75       200
```

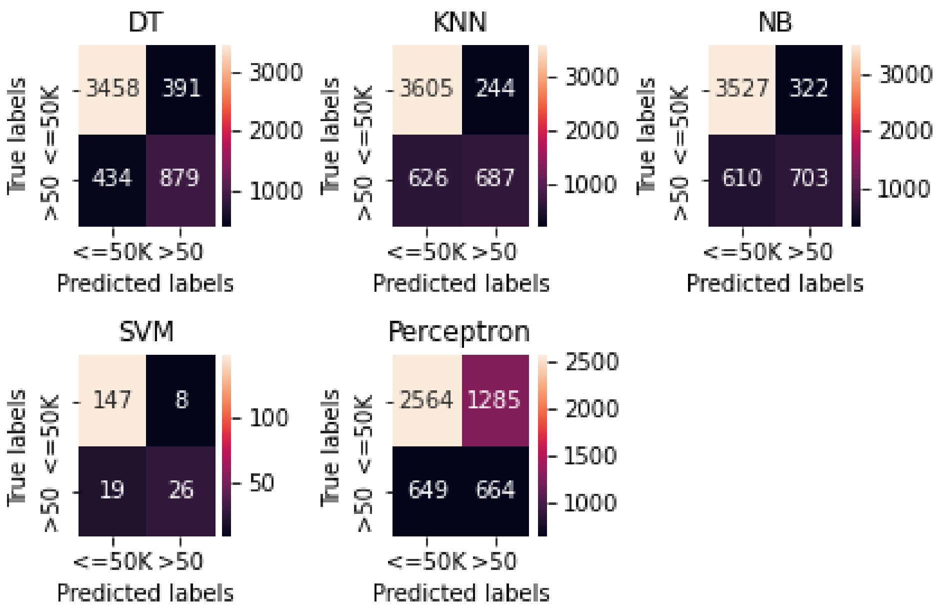
Figure 5: Perceptron evaluation

```
              precision    recall  f1-score   support

           0       0.80      0.67      0.73      3849
           1       0.34      0.51      0.41      1313

    accuracy                           0.63      5162
   macro avg       0.57      0.59      0.57      5162
weighted avg       0.68      0.63      0.64      5162
```

Figure 6: CM before turning



Except the Perceptron classifier the other classifiers are showing similar performance in terms of accuracy ranging from 83 to 76 and a f1 score for the minority (positive) class from 59 and 61.

The Perceptron classfier is showing the worst performance with an accuracy of 63 and an f1 score of 63 for the positive class. It is able to classify only 51% of the instances belonging to the minority class and is making a with a precision of 34% a lot of false positive predictions.

### Model parameter tuning

*Use the validation set to find the best parameters for each classifier. Report on the tuning process and on the chosen parameters for each model.*

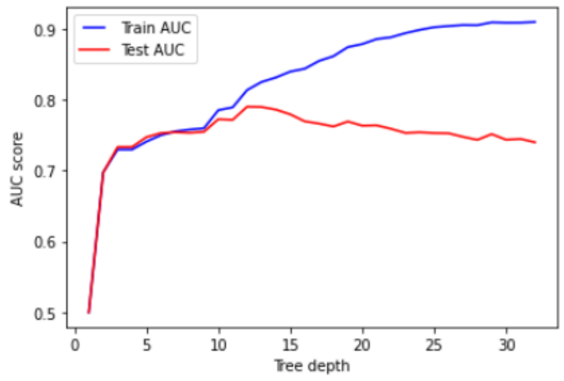
For the DT and KNN models we decided to test all possibles parameter combinations with the so called grid search approach for which the sklearn library is already providing methods. The grid search method is building a matrix with a combination of all parameters we want to test. As evaluation setup we specified to use 10 fold cross validation with stratified K-Folds. First we decided which parameter we want to try out different values.

For the **DT classfier** we decided to test all combinations of the following parameters

$Max\ depth = \{1, 3, 5, 10, 20, 30\}$
$Min\ sample\ split = \{0.01, 0.02, 0.03, 0.04, 0.05\}$
$Splitting\ criterion = \{"gini", "entropy"\}$

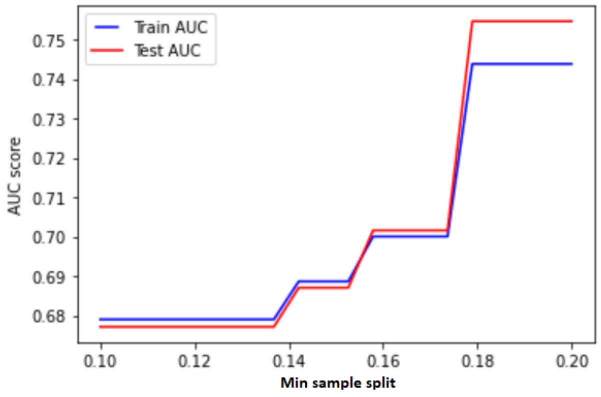
The parameter Max depth limits the maximum depth of the Tree. Here is a ROC curve for the dept of the DT to help us visualize its behavior.

Figure 7: Train and test performance of DT classfier with max depth param



Here is a ROC curve for the sample split size of the DT to help us visualize its behavior.

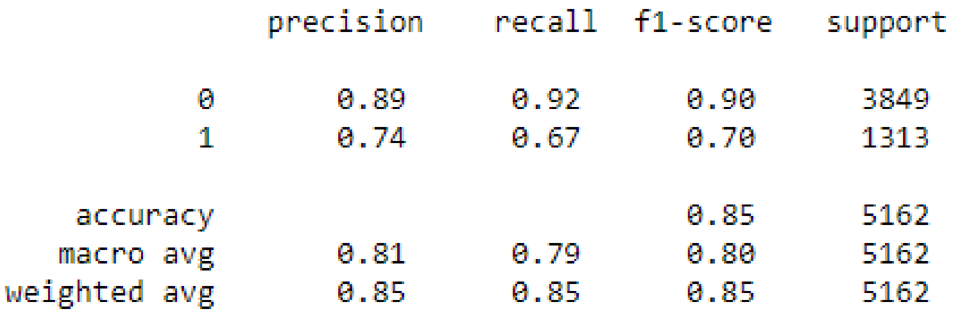
Figure 8: Train and test performance of DT classfier with min sample split param



The parameter Min sample split defines the minimum number of samples each internal node of the tree can contain in percentage of the data.
So we tested 60 parameter combinations in total. Choosing the **best** parameters depends on the **Task**. As a main difficulty with this data set is the class imbalance we assumed that the task is to correctly classify as many of the instances belonging to the minority class. Because of this we optimized our parameters for the best f1 score for the minority class.

Figure 9 shows the values for the different performance metrices for the DT model after parameter tuning. With this setting we were able to reach 1% improvement for the f1 score of both classes and the accuracy of the DT model. Although this is a small improvement in accuracy and and f1 scores we improve the models precision for predicting instances from the minority class (The positive class) with 74% precision. 5% improvement with the parameter tuning. The recall rate for the minority class stays the same.

Figure 9: DT performance after parameter tuning



```
              precision    recall  f1-score   support

           0       0.89      0.92      0.90      3849
           1       0.74      0.67      0.70      1313

    accuracy                           0.85      5162
   macro avg       0.81      0.79      0.80      5162
weighted avg       0.85      0.85      0.85      5162
```

The best parameter for the DT classifier are:

- Splitting criterion: Entropy
- Max depth: 15
- Min sample split: 0.01

This meas that our DT classifier is quite deep and internal (non leave) nodes of the tree can contain less number of samples. For the train set with cross validation each internal node can contain minimum 255 samples. This is an indicator that our model may memorizes too much.

For the **KNN classifier** we selected the following parameters:

$n\ neighbors = \{1, 3, 5, 8, 10, 15, 20, 30\}$Number of neighbors to use.
$p = \{1, 2\}$ Power parameter for the Minkowski metric.

To visualize the behavior of the KNN Agorithm for different K we can utilize the ROC-Curve:

Figure 10: Train and test performance of KNN classfier with n-neighbors param

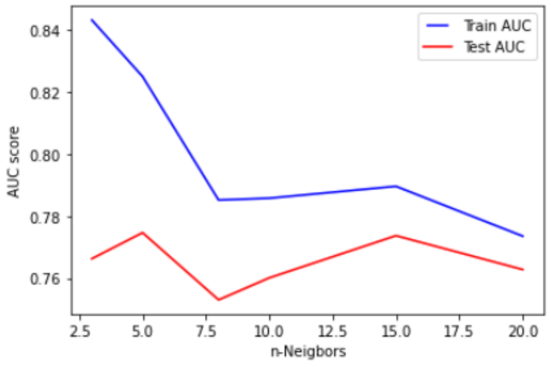
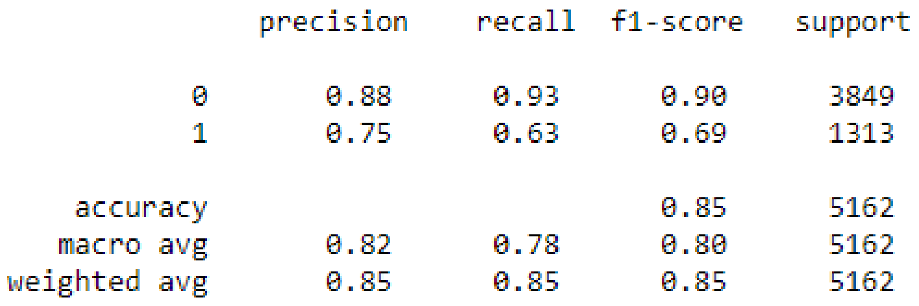


Figure 11 is showing the test performance for the KNN classifier applying the best parameters we found applying the grid search method.

After the parameter tuning the KNN classifier is showing 2% improvement in accuracy 1% improvement for the f1 score of the majority class (negative class) and a bigger improvement of 5% for f1 score of the minority class (positive class). The recall rate for the minority class increased from 52% to 63% and the precision from 74% to 75%.

Figure 11: KNN performance after parameter tuning



```
              precision    recall  f1-score   support

           0       0.88      0.93      0.90      3849
           1       0.75      0.63      0.69      1313

    accuracy                           0.85      5162
   macro avg       0.82      0.78      0.80      5162
weighted avg       0.85      0.85      0.85      5162
```

The best parameter for the KNN classifier are:

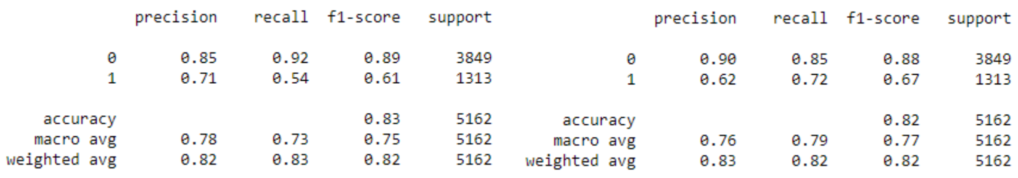
- n neighbors: 15
- p: 1

For tuning the **Naive Bayes** classifier we tested two different implementation of the Naive Bayes classifier.

- The Gaussian Naive Bayes implementation which is as of our understanding better suitable for features with continuous values.
- The Bernoulli Naive Bayes implementation which is better for features which can take binary values.

In addition we tested these two classifiers with and without sklearns class weight method wit the *balanced* option. This method is weighting the samples in such way that the minority class samples are replicated until having as many samples as in the majority class.

Figure 12: NB performance after parameter tuning (left: Gaussian w/o class weight) (right: with class weight)



```
              precision    recall  f1-score   support

           0       0.85      0.92      0.89      3849
           1       0.71      0.54      0.61      1313

    accuracy                           0.83      5162
   macro avg       0.78      0.73      0.75      5162
weighted avg       0.82      0.83      0.82      5162
```

```
              precision    recall  f1-score   support

           0       0.90      0.85      0.88      3849
           1       0.62      0.72      0.67      1313

    accuracy                           0.82      5162
   macro avg       0.76      0.79      0.77      5162
weighted avg       0.83      0.82      0.82      5162
```

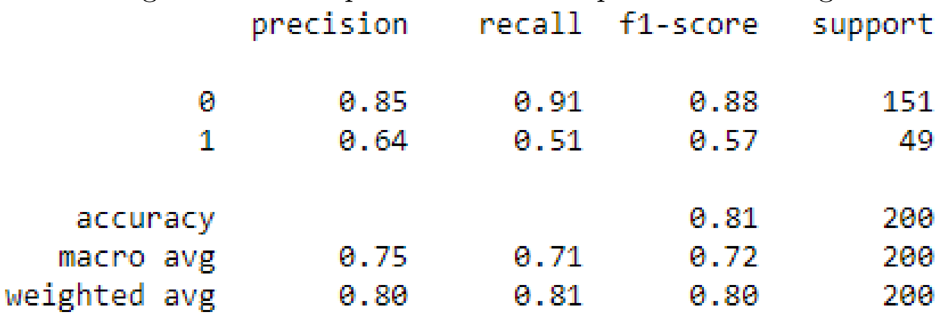
For the Naive Bayes classifier one can decided (depending the Task to solve) between two classifier where the f1 score for the minority class predictions improved. Figure 12 is showing the performance metrices for Gaussian Naive Bayes without balanced class weights on the left side and with weighting on the right side.

One is improving the f1 score for the minority class by 1% from 60 to 61 the other one improves the f1 score by 7%. But the model with lower f1 score has higher precision (71%) while the other has a higher recall rate (72%).

For the **SVM classifier** we selected the following parameters:

$kernel = \{'linear', 'poly', 'rbf', 'sigmoid'\}$
$regularization = \{0.4, 0.6, 0.8, 1\}$

Figure 13: SVM performance after parameter tuning



```
              precision    recall  f1-score   support

           0       0.85      0.91      0.88       151
           1       0.64      0.51      0.57        49

    accuracy                           0.81       200
   macro avg       0.75      0.71      0.72       200
weighted avg       0.80      0.81      0.80       200
```

Testing the SVM classifier was challenging for of performance reasons. Although we trained the model with a subset of the dataset (1000 samples) the training process took very long time.

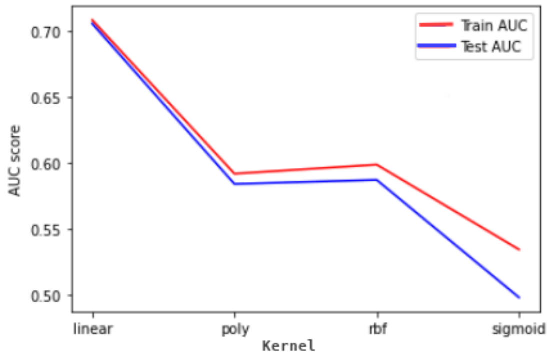
With the SVM classifier we were able to improve the negative class f1 score from 83 to 88 and the accuracy from 76% to 81 but were not able to improve the f1 score of the minority class. It decreased from 59 to 57. The recall rate dropped from 57% to 51%

but the precision increased fro 64% to 66%. The best parameter for the SVM classifier are:

- Regularization: 0.8
- Kernel: linear

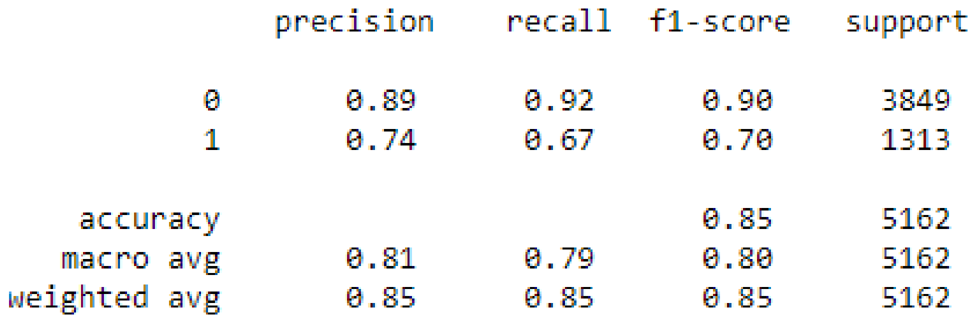
Here a Plot on how the other Kernels performed:

Figure 14: Train and test performance of SVM classfier with kernel param



For the **Perceptron** classifier we tested different values for learning rate alpha parameter $alpha = \{0.00001, 0.0001, 0.0002, 0.001, 0.002, 0.05\}$. The best performance we saw with the lowest learning rate.

Figure 15: Perceptron performance after parameter tuning



```
              precision    recall  f1-score   support

           0       0.89      0.92      0.90      3849
           1       0.74      0.67      0.70      1313

    accuracy                           0.85      5162
   macro avg       0.81      0.79      0.80      5162
weighted avg       0.85      0.85      0.85      5162
```

With changing the learning rate parameter the Perceptron classifier showed a significant performance improvement:

- 22% improvement in accuracy
- 17% improvement of the f1 score of the majority class
- **29% improvement for the f1 score of the minority class**

**Model interpretation/visualization**

For the Decision tree classifier we plotted the decision tree graph (Figure 16). The majority sample class is encoded in the color and the number of samples (how many percent of the entire data set) is encoded in the color intensity. Right edges are answering the splitting variable with False while left edges with True. The root node is splitting for the feature "Marital status" having the value Maried civ spouse or not. If a person is not married, the entropy drops from 0.80 to 0.36.

Notable is that the tree size is quite big (deep and also broad) what makes it hard to interpret. We tried some different approached to earlier stopping the the tree from growing, but couldn't find a smaller tree where the performance for the f1 score of the minority class does not drop too much.

Figure 17 is showing a bar plot for the feature importance of the decision tree. The top 5 important features for which we see the most drop in impurity are the features (*Marital Status, Education num, capital, age and hours per week.* For the numerical features *Education num*, *age* and *hours per week* we saw already high positive correlation values in the correlation matrix. For the categorical features we performed chi square test and noted that all categorical features shows correlation to the income. For the Marital Status feature we also saw in our plots that married people are more likely to earn more then 50K. Therefore the feature importance for these variables ailing with our data analysis and is quite reasonable for us.

Figure 16: Decision tree visualizing all nodes

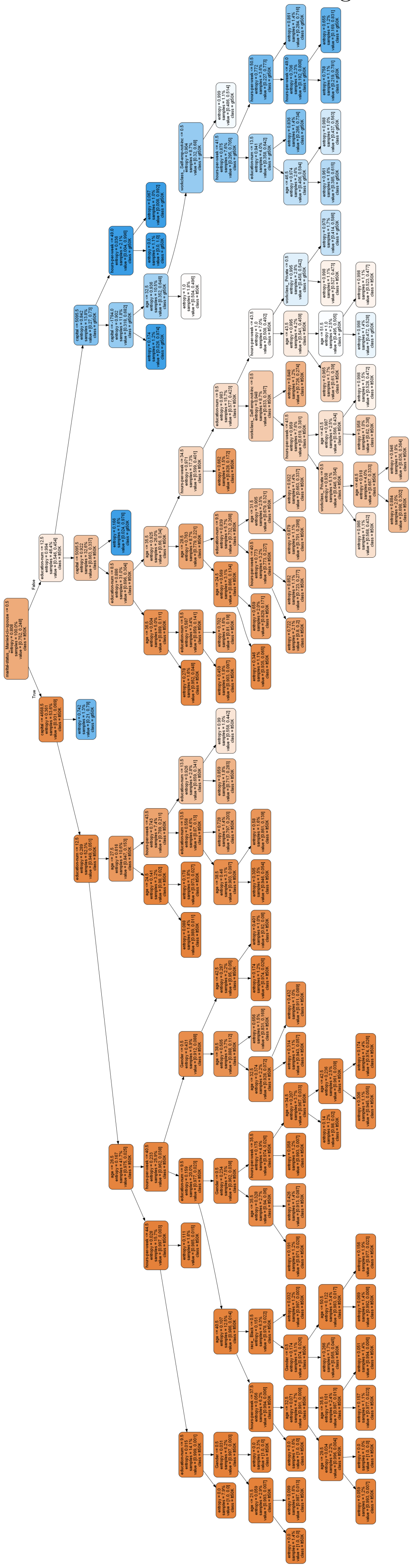
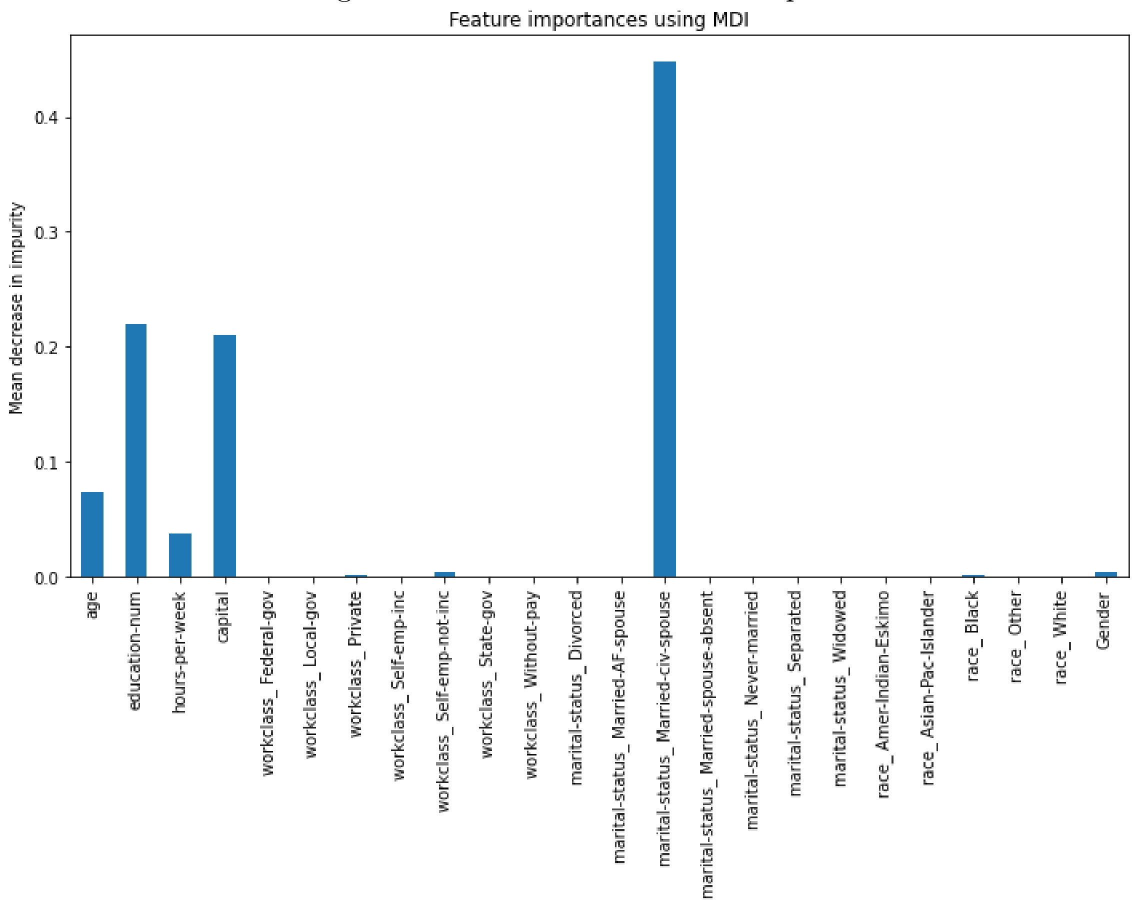


Figure 17: Decision tree feature importance



We also plotted some projected scatterplot for points in the 3 best transformed dimensions using a Principal Component Analysis. That generated pretty pictures, but those pictures are not really useful to us. For example without the Captions it would be nearly impossible to even tell which of the 2 following Plots plots the decision boundary of a Decision Tree and which plots the decision boundary of a Naive Baysian Classifier.

Figure 18: Naive Baysian Clasifier Decision Boundary

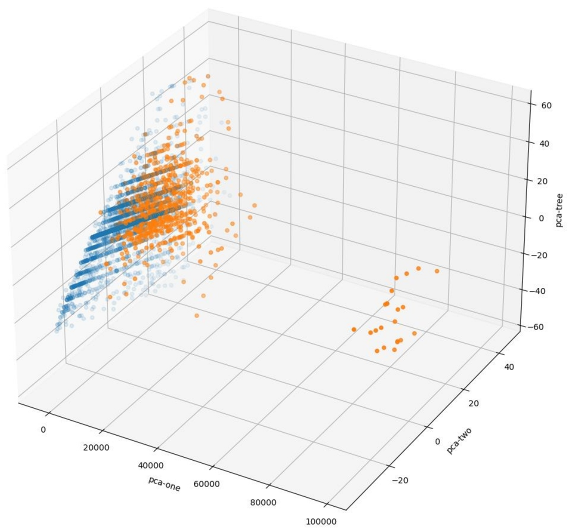
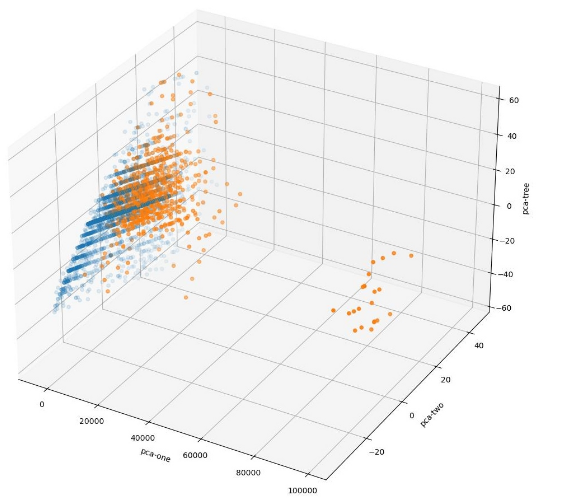


Figure 19: Decision Tree Decision Boundary



Therefore while we can try to visualize it this way, we found it not helpful and will not further use this Method for this specific dataset.

## Effect of model parameters

*For each classifier, list the parameters that affect its performance. For each classifier, choose at least one such parameter3 and experiment with its effect on model performance (both training and testing errors). Report on the results, also with regard to overfitting.*

For the DT classfier the max depth and the min sample split size parameters are affecting the performane of the DT.

Figure 7 shows is showing the AUC values for the train and Test run with different values for the tree depht. It is notable that test performance stops improving around the tree depth 13 while the AUC score for the train runs still improving.

The AUC curves for the *min sample split size* parameter of the DT classifier (see Figure 8) is showing that the test performance is getting better for values bigger then 0.16 while the train performance is decreasing.

Figure 10 is showing the AUC curve for the n-neighbors parameter of the KNN classifier. It is notable that the test performance has two peaks. One at n=5 and another at k=15. The train AUC score also starts to decrease after k=15.

For the SVM classifier we plotted the AUC score for train and test performance for the 4 different kernel types (See Figure 14). It clearly notable that the linear kernel has the best performance for train and test run. For the other kernel types we see a slightly better performance for the training. So it seems to generalize not so good to the test data.

## Discriminative behavior

We will use our Decision Tree Classifier because it performed best to test for discrimination. The Code can be found in the file "Discriminative.ipynb". Let us first calculate the different measures from lecture 6 to determine how bad it is using a testset with 5162 members.

```
Number of males, that earn more than 50K: 922
Number of females, that earn more than 50K: 124
Number of people, that earn more than 50K: 1046
Ratio between +50K/All poeple: 0.2026346377373111
If a person is male: 0.2637299771167048
If a person is female: 0.0744297719087635
```

So we can clearly see that the Statistical parity is badly broken and man are 4 times more likely to be predicted over 50k than Woman.

```
Number of males, with income >50k and predicted income matches: 704
Number of females, with income >50k and predicted income matches: 95
Number of people, with income >50k and predicted income matches: 799
Error ratio of False Negative Predictions: 0.39146991622239147
If a person is male he will be wrongly classified with: 0.3719892952720785
If a person is female she will be wrongly classified with : 0.5052083333333334
```

As we see woman get missclasified in 50% of cases while Males get only missclasified in 37% of cases. So we also fail Equal Opportunity.

Finally let us check for Disparate Mistreatment

```
    DM_female = np.abs(FNR_female) + np.abs(FNR_female)
    print("Difference in model's prediction errors between ...
    ... protected females and non-protected groups:", DM_female)
    Difference in model's prediction errors between protected ...
    ... females and non-protected groups: 0.21858339680121863
```

So the Disparate Mistreatment is confirming the first to measures, the Classifier is discriminating Woman.

**How would you mitigate classifier discrimination? Briefly outline your idea(s):**

One large Problem which we already identified in 3.1 is that we have far fewer data about Woman than about Man.to have Parity we need 2x more Woman and 6x more Woman making >50k

For the KNN-Classifier we can try duplicating the data we have for the woman over 50k 6 times to reach statistical parity between male and female datapoints over 50k so that the classifiers compares woman now equally to other woman making over 50k and not disproportionally to man making over 50k, which might help.

For The Decision Tree we could manually force the root to be the attribute "sex" and then train 2 separate Decision Trees, one for Man and one for Woman. That way we can tune the parameters separately to at least reach Equal Opportunity by making sure both main branches have the False Negative and False Positive errors of same magnitude.

Witch this method we could also instead of training 2 Decision Trees train 2 SVMs, KNNs, etc and again tune the parameters of both branches to ensure Equal Opportunity.

## Outperforming Libraries

We implemented the KNN Algorithm. The Code can be found in the file "ProjectKNN.ipynb". First we import and prepare the Data according to 3.1, then we normalize al data to be between -1 and 1, and then we implement the KNN Algorithm. Our implementation uses numpys vectormath CPU instruction to keep the implementation from taking forever to finish. The important lines are the distance functions. Those are:

```
difference = numpy.subtract(mat_a,vec_b)**2
difference = difference.sum(axis=1)
```

We also checked the L-1-Distance computed by:

```
difference = numpy.subtract(vec_a, vec_b)
distance = numpy.linalg.norm(difference, axis=1)
```

End finally we will try the naive "stupid" approach for the KNN using 2 nested for loops using:

```
difference = [0]*len(mat_a)
```

```
    for i in range(len(mat_a)):
        difference[i]=0
        for j in range(len(vec_b)):
            difference[i] = ( mat_a[i][j]-vec_b[j] )**2 + difference[i]
    return numpy.sqrt(difference)
```

Finally we compared the Runtimes and Accuracy's we got to the KNN-Algorithm from sklearn which we easily trained and executed using:

```
sklearnknnclass = KNeighborsClassifier(n_neighbors = k)
sklearnknnclass.fit(adultknndata, adultknnlabels)
skscore = sklearnknnclass.score(adultknn_testdata, adultknntestlabels)
```

So here are the Results we got. Be aware that runtime is highly depended on the computer you run the program on so if you execute the "ProjectKNN.ipynb" yourself you will get different times than i on my old Thinkpad T420 from 2012. Because that poor old laptop already struggled with only 5% testdata we left it at that. So 95% of the adult dataset are as points and we go through 5% not yet known to the classifier and test those. This means the size of trainingset is 32561 instances and Size of testset 1628 Instances. That is more than enough to show performance differences. Also Note that we randomly select a sample of 5% of the dataset to be our testset, so for different randomly selected test sets we also get different accuracy's. In our experiments both accuracy and time varied by 15 percent.

| **Algorithm** | Euclіddistance | L1-Distance | Sklearn Library | nested for-loops Euklid |
|---|---|---|---|---|
| K=1 Runtime | 88.727s | 68.76s | 0.608s | - |
| k=1 Accuracy | 94.04 % | 94.04% | 94.04% | - |
| K=3 Runtime | 81.03s | 82.337s | 0.63s | 612.364s |
| k=3 Accuracy | 89.8% | 89.8% | 89.68% | 89.8% |
| K=5 Runtime | 80.504s | 97.434s | 0.676s | - |
| k=5 Accuracy | 87.71% | 87.71% | 87.84% | - |
| K=9 Runtime | 123.803s | 89.416s | 0.712s | 835.582s |
| k=9 Accuracy | 86.43% | 86.43% | 86.36% | 86.43% |
| K=13 Runtime | 82.123s | 95.492s | 0.761s | - |
| k=13 Accuracy | 85.32% | 85.32% | 85.57% | - |

**So what do we Observe?**

- Using nested for loops for the KNN is absolutely terrible we therefore have only done that twice because sitting around for 10 minutes is boring.

- Te accuracy is everywhere roughly equal, which is not surprising since all KNN implementation should do the same. Interestingly it did not make any big difference whether we used L1 distance or Euclidean which is surprising.
- Both the KNN using Euclidean and L-1 Distance took roughly the same amount of time to run, (≈85s) with one massive outlier of 123s which was probably caused by background tasks.
- Performance wise its: Libraries are with ≈0.8s 100x faster than the smart way which is 10x faster than the naive way which takes over 10 minutes.

**Conclusion:** You cant compete with highly optimized precompiled low level C-Code that uses dark magic like Bitflipping and Registershifting to squeeze out the last bit of performance using a interpreted high level language like Python. Just use the Libraries!