Machine Learning, Fall 2019: Project 3
out on 10/15/19 - due on 10/29/19 by noon on Blackboard

Anastasija Mensikova

Header: Resources used for the completion of the project:

- Python programming language
- Jupyter Notebook
- Scikit-Learn ML Library for Python
- SVM Scikit examples:
  https://scikit-learn.org/stable/auto_examples/svm/plot_svm_margin.html
- Information Gain Library:
  https://pypi.org/project/info-gain/
- Lecture slides
- MatPlotLib for plots

1 Project Description

This project implements the support vector machines (SVM) to solve the Adult Census Income kaggle task: https://www.kaggle.com/uciml/adult-census-income to predict whether income exceeds $50K/yr based on 1994 census data. However, instead of using the complete dataset, we are only using 10% of it in order to save time during training.

Support Vector Machines (SVMs) are a type of supervised learning that can be used both for regression and classification. They are very effective for datasets with high dimensions, and the use of kernels in SVMs are generally efficient at improving the performance of the classification or regression. However, the time that it takes to perform classification with SMVs and high risk of overfitting are causing more and more people to steer away from them. It is, however, very interesting and important to perform experiments for high-dimensional datasets using SVMs for the sake of testing, and this is what the goal of this project is. The full project can be found in the attached Jupyter Notebook.

2 Dataset preprocessing and interpretation

1. **5 points** Replacing the '?'s.

   There are multiple ways to get rid of questions marks (or NA values) in a dataset. Since I loaded the dataset into a dataframe, rather than just a simple list or a dictionary, using existing Pandas methodologies was the most intuitive thing to do. Upon examining all the features, it was apparent that the only features that contained '?' values were categorical/nominal values rather than numberic. For that reason, instead of calculating the mean and replacing the '?'s with the mean of each feature, the mode (or the value occurring most often) was used as a replacement value. The following code snippet shows how it was done for the *workclass*, *occupation*, and *native_country* features (since they were the only ones containing the '?'s).

```
most_often = df.mode(dropna=True)
for column in columns_with_question:
    df[df[column] == "?"].apply(lambda x: most_often[column][0])

df.loc[(df.workclass == '?'), "workclass"] = df.mode(dropna=True)['workclass'][0]
df.loc[(df.occupation == '?'), "occupation"] = df.mode(dropna=True)['occupation'][0]
df.loc[(df.native_country == '?'), "native_country"] = df.mode(dropna=True)['
    native_country'][0]
```

2. **5 points** Dealing with discrete (categorical) features.

   Although dealing with categorical values can be done in numerous ways, for the sake of this project Pandas *factorize* method was used to first factorize all the nominal values, and then min-max rescaling was using to rescale all the values to be within the 0 to 1 range to make the job a little simpler for the SVM. However, those values were added in new columns instead of overwriting the existing ones to preserve those for the future. Pnadas *factorize* method helps to get the numeric representation of an array by identifying distinct values, and it appears to be the most efficient way to perform this operation. The exact code for how it was done can be found in the Jupyter notebook.

3. **5 points** Stratified 10-fold-cross validation.

   Even though in the previous project Stratified K-Fold validation was implemented from scratch, due to the use of Scikit-Learn library in this project, its very own Stratified K-Fold validation split method was used, which is very helpful in ensuring the correctness of the split performed. The code for how it was done can be found below.

```
def cross_validation_num(x, y):
    skf = StratifiedKFold(n_splits=10, shuffle = False, random_state = 2)
    data = []
    for train_index, test_index in skf.split(x, y):
        x_train, x_test = x[train_index], x[test_index]
        y_train, y_test = y[train_index], y[test_index]
        data.append((x_train, x_test, y_train, y_test))
    return data

```

4. **5 points** Features with highest information gain.

   Information gain for each feature was calculated using the Information Gain python library linked above, and the exact code for that can be found in the attached Jupyter Notebook. After performing such calculations fnlwgt and relationship were identified as features with highest information gain, and the scatter plot of such can be be found in Figure 1. However, a separate heatmap was built to show the general relation between all features in the dataset. Although this wasn't used for feature selection based on information gain, it is interesting to note that features such as age and education seem to have very high impact on the income level. The heatmap discussed can be found in Figure 2.

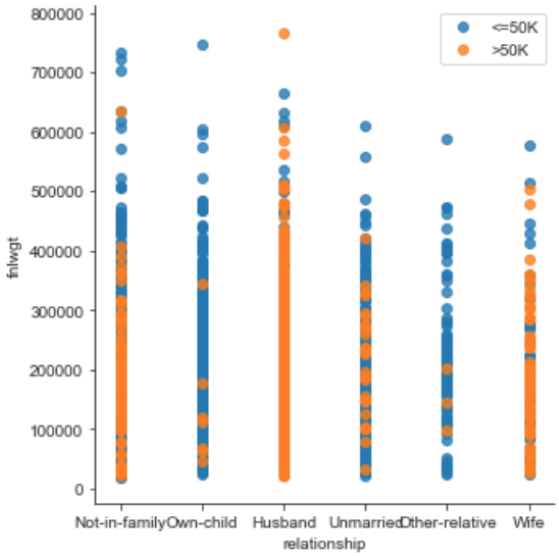

Figure 1: Highest Information Gain Scatter Plot

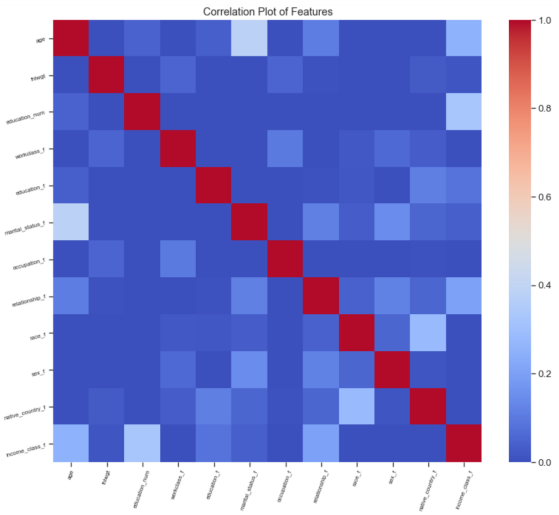

Figure 2: All Features HeatMap

3 Using a linear soft-margin SVM

1. **15 points** Train your SVM with *stratified 10-fold-cross-validation* on the 2 features with the highest information gain and visualize your boundary.

   As always, the entire code for all of the SVM runs can be found in the Jupyter Notebook attached. For the sake of the very first test, the SVM with top two features was run with the value c=0.1. Achieving the best score of 0.76 and the best F1-measure of 0.66. The visualisation of the SVM decision boundary for this example can be found in Figure 3 below.

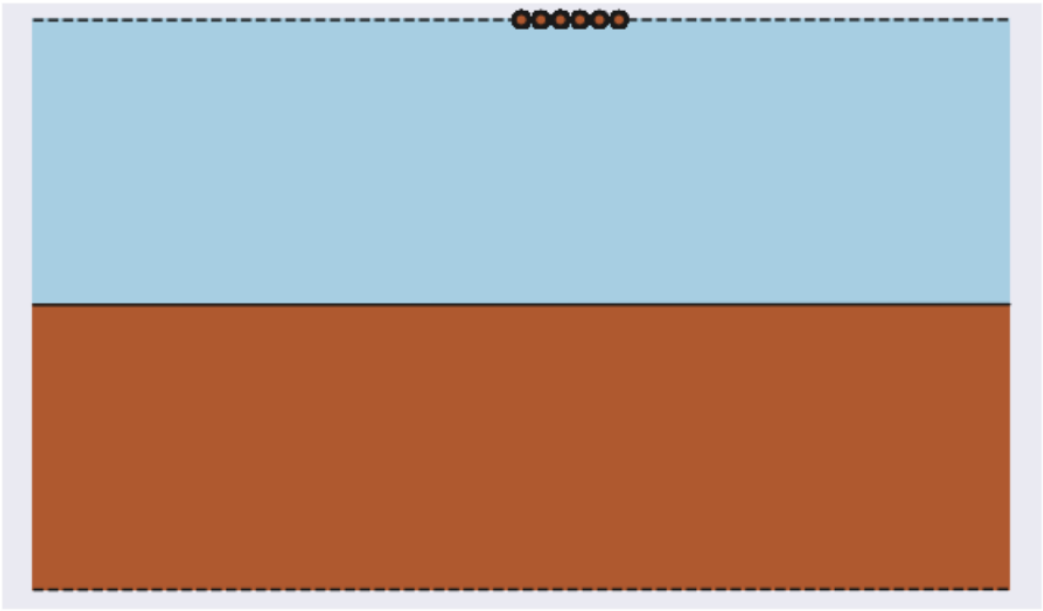

Figure 3: SVM Decision Boundary

2. **15 points** Change the hyper-parameter C from small to larger values.

   As it can be observed in the results portrayed in the Jupyter Notebook, increasing the value of c to much larger values (of 100 and 1000) does not seem to make any major differences - good or bad. However, a larger c, given Figure 4 below, does seem to allow for a larger amount of samples on the margin.

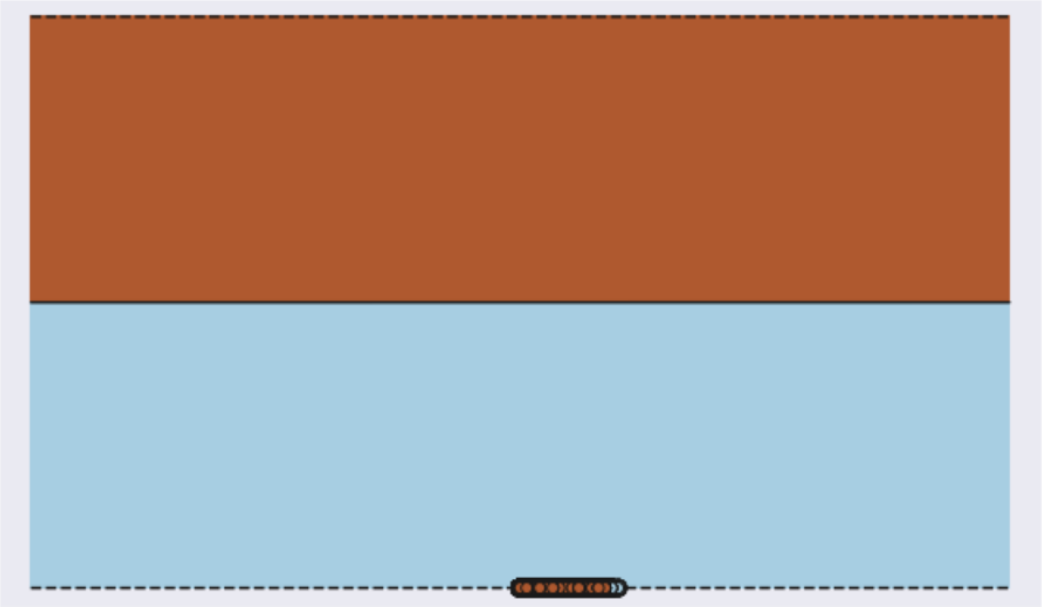

Figure 4: SVM Decision Boundary for c=100

3. **15 points** Train the SVM using all the features.

   As it can be seen in the code, using all features does appear to increase both the score and the F1-measure quite dramatically, with the best score fluctuating up to 0.82. The following array of c values was taken into consideration when determining the most optimal c value: [0.001, 0.1, 5, 10, 100] and score for each was compared when trained on all features. It was found that the c value of 10 provided the most optimal results with the score of 0.79. The full code and the methodology can be found in the Jupyter Notebook.

4 Using SVM with a kernel

1. **15 points** Compare the performance (precision, recall, f1-score, and variance) of different kernels: Linear, RBF, and polynomial.

   Linear

   - Score: 0.7945962542216763
   - Average precision score: 0.36302879382148767
   - Precision: [0.24101934 0.68238994 1.]
   - Recall: [1. 0.27643312 0.]
   - F1: 0.7599769852992084
   - Variance: -0.010473481334899892

   RBF

- Score: 0.7589806570463616
- Average precision score: 0.2410193429536383
- Precision: [0.24101934 1.]
- Recall: [1. 0.]
- F1: 0.6549834820103355
- Variance: 0.0

Polynomial

- Score: 0.7589806570463616
- Average precision score: 0.2410193429536383
- Precision: [0.24101934 1.]
- Recall: [1. 0.]
- F1: 0.6549834820103355
- Variance: 0.0

It is obvious that the linear kernel seems to have a slightly better performance for this specific dataset, whereas the RBF and Polynomial kernels seem to only have slight differences to them in terms of performance.

2. **20 points** Try your best to get higher performance!

   Although there is a huge variety of ways the performance of SVM could be improved, for this particular project I really wanted to experiment with using one-hot encoding as it is known to provide great performance improvements. The experiments with one-hot encoding were performed with different c values as well as with top features and all features. One-hot encoding was implemented with the help of the Scikit-Learn library's own encoding, as it can be seen in the code, and it seemed to provide slight improvements to the overall performance of the SVMs. However, given the heatmap, using only the Age and Education features (as they affect the income the most in the heatmap), provides the best results overall for a variety of c values. The score of such reaches 0.82.