Final Project: Salary Prediction

Mary Wang, Xuanyi Li, Yangfan Wang

Summary

We conducted a study to explore the performance of various machine learning models in predicting an individual's salary based on their demographic and socio-economic attributes. The dataset used in this study was obtained from the UCI Machine Learning Repository and contained information on over 30,000 individuals. We evaluated the performance of four different models - K-Nearest Neighbors, Logistic Regression, Decision Trees, and Support Vector Machines - using four evaluation metrics: misclassification rate, sensitivity, specificity, and F1 score. We also used permutation importance to determine the most important features for each model. Our results indicate that each model has its own strengths and weaknesses in predicting salaries, with different features having varying degrees of importance in each model. We discuss the limitations of our methodology and highlight the need for further research to explore the socio-economic and cultural factors that may impact salaries.

Introduction

Dataset

In this project, we aim to apply classification methods to analyze the Salary Prediction dataset. The dataset used is obtained from the UCI Machine Learning Repository(Kohavi and Becker). The dataset includes information about individuals from diverse backgrounds and their income levels, specifically whether their income exceeded $50K per year, as recorded in a census conducted in 1994. It consists of 32,562 observations, with 14 features describing various attributes of the individuals such as age, education, occupation, and marital status, and two classes indicating whether their income exceeded $50K. The dataset consists of the following features:

- **age**: age of an individual; integer between 17 and 90.
- **workclass**: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
- **fnlwgt**: final weight, the number of people the entry represents; positive integer.
- **education**: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
- **education-num**: the highest level of education achieved; integer between 1 and 16.
- **marital-status**: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
- **occupation**: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
- **relationship**: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
- **race**: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
- **sex**: Female, Male.
- **capital-gain**: capital gains for an individual; non-negative integer.

- **capital-loss**: capital loss for an individual; non-negative integer.
- **hours-per-week**: hours an individual works per week; integer between 1 and 99.
- **native-country**: country of origin for an individual.

Research Problem

The primary objective of our project is to apply various classification methods to the Salary Prediction dataset and achieve the following two research problems:

1. To compare the performance of various classification methods using metrics such as misclassification error, confusion matrix, training error, etc. The goal is to identify the most suitable model for accurately predicting an individual's income level based on the features provided.
2. To identify the most significant features that influence income levels and summarize their effects. By analyzing the impact of different characteristics on income levels in various models we train, we can provide insights into the factors contributing to income inequality.

Methodology

Data preprocessing

Before proceeding with the classification analysis, we preprocess the dataset to ensure its suitability. The following changes are made:

- **Check for missing data**: Notice that there are 583 null values for the feature 'native- country' and 1,836 null values for 'workclass' making up 1.9% and 5.6% of all data respectively. Generally there are two methods to handle missing values in a dataset, namely deleting the rows with missing data or replacing the null values with either the mean, median, or mode of the feature. In our case, both features are categorical, deleting rows will be a better fit. Plus, we examined the distribution of `salary` for rows with null values and confirmed that deleting those rows will not introduce significant bias. Thus, we decided to delete those rows and resulted in 30,162 rows in total.
- **Delete columns**: We first deleted column `fnlwgt` because it is the weight of each row in the dataset. Therefore it cannot be used as a feature. In addition, we deleted the column `education` which is a duplicate of `education-num`.
- **From categorical to numeric**: The package 'Scikit-learn' requires categorical features to be encoded as numbers, rather than strings like 'female'. There are two types of encoding for categorical data: numeric encoding and "one-hot" encoding. In numeric encoding, each unique category value is assigned an integer value. For example, "red" is 1, "green" is 2, and "blue" is 3. In one-hot encoding, "Each bit represents a possible category. If the variable cannot belong to multiple categories at once, then only one bit in the group can be 'on'.", said Zheng and Casari(2018). In general, numeric encoding is used when there is an inherent order of the categories, and one-hot encoding otherwise. In our case, we do not have any inherent order in any of our features. Hence, we applied 'one-hot' encoding to all of our features except `occupation` and `native-country`. We numerically encoded `occupation` and `native-country` because both of them contain more than 10 categories which otherwise will significantly increase the dimensionality of our dataset and make our models prone to overfit.
- **Prepare training and test data**: We shuffled the data and divided it into 80% for training and 20% for testing where training set is used for model training and testing set is used for model evaluation for result reporting.
- **Deal with imbalance data**: In the final step of data preprocessing, we examined the distribution of our responses and noticed that there is a significant imbalance between two possible salary levels(75% for `<= 50K` and 25% for `>50K`). When a dataset is imbalanced, a model will tend to learn more about

the majority class, and as a result, the model's performance will be biased towards that class. This can lead to low precision, recall, and F1 scores on the minority class. Hence, we resampled training data with replacement and achieved an almost balanced training set.

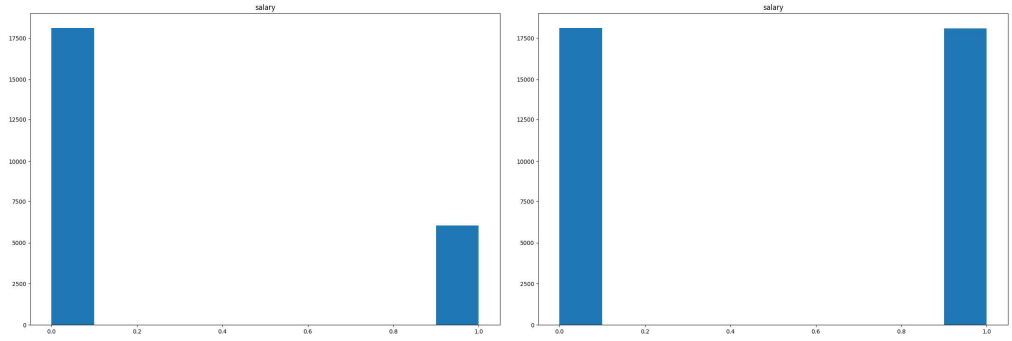

Figure 1: Salary Distribution Before and After Resampling(1: >50K; 2: <= 50K)

- **Exception for KNN**: The resampling method above should not be applied to the KNN model. The reason for this is that KNN is a non-parametric algorithm, which means that it does not learn a model based on the data but instead directly memorizes the training examples. Resampling techniques can change the distribution of the training data, which can affect the distance metric used by the KNN algorithm.

Models

We applied four classic classification models, namely K-Nearest Neighbors, Logistic Regression, Support Vector Machine and Decision Tree. For each model, we explored various parameters and hyperparameters, and used the Exhaustive Grid Search technique to determine the optimal model for hyperparameter optimization. This involves testing every possible combination of the hyperparameters and determining the best values through cross-validation(k = 5, scoring='accuracy'). We used libraries `pandas`, `category_encoders`, `numpy`, `matplotlib.pyplot`, `seaborn`, `random`, `sklearn`, `matplotlib.pyplot` in our analysis. The following are the parameters we will tune for each model:

- **K-Nearest Neighbors**: n_neighbors:[1-50]; weights:['uniform', 'distance']: check whether adding weights to the data points is beneficial to the model or not.
- **Logistic Regression**: penalty:['l1', 'l2', None]; C: inverse of regularization strength. smaller values specify stronger regularization: [1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03]; solver: algorithm to use in the optimization problem: ['lbfgs', 'liblinear', 'saga']
- **Support Vector Machine**: penalty: [0.1,1,10]; kernel: ["rbf", "poly", "sigmoid"]; degree: degree of the polynomial kernel function [2, 3]; gamma: kernel coefficient for 'rbf' ["scale", "auto"].
- **Decision Tree**: criterion: the function to measure the quality of a split; max_depth: ['gini', 'entropy']; max_dept: [5, 10]; min_samples_leaf: [100, 500]; min_samples_split: [100, 500]; max_features:[None, 'sqrt'].

After obtaining the best combination of hyperparameters, we trained each model using the entire training dataset and predicted the test data. Then we compared the performance of each model using various metrics such as accuracy, sensitivity, specificity, F1 score and ROC curve. We also analyzed the performance of the models and identified features contributing most to the performance of a model. For instance, if a certain feature has an overall high ranking in terms of the importance in all models, this may indicate its significance in predicting the correct outcome.

Although our methodology is similar to standard practices for preprocessing and modeling in machine learning, the specific details of our approach, such as the choice of resampling technique and the parameter tuning for each model, may differ from what others have done in the past. One key difference from existing methodologies is the resampling step to deal with imbalanced data. This step helps to mitigate the potential bias that can occur in the model's performance due to the imbalanced distribution of the response variable. Moreover, the Exhaustive Grid Search technique for hyperparameter optimization is a widely used method, but the specific parameters we have chosen to tune for each model may differ from other methodologies. Finally, the evaluation metrics we use differ from others. While most analysis only considers accuracy, we take a closer look at our model performance within each class to ensure a balance between sensitivity and specificity.

Results

KNN

By analyzing the CV plot, we observe that the accuracy is lower when the number of neighbors k is less than 10. This is because a smaller k value implies a more complex model, resulting in a larger variance and overfits the data. It is an illustration of the bias-variance tradeoff. When k is between 15 to 50, the accuracy is quite similar. Another important hyperparameter is the voting method. Majority voting gives equal weights to every neighbor, while distance weighting assigns less weights for neighbors with greater distances. In our case, majority voting performs better than distance weighting for all k, but the difference is less than 0.1 in terms of accuracy. The optimal hyperparameter for our KNN model is k = 28 and weights = uniform.

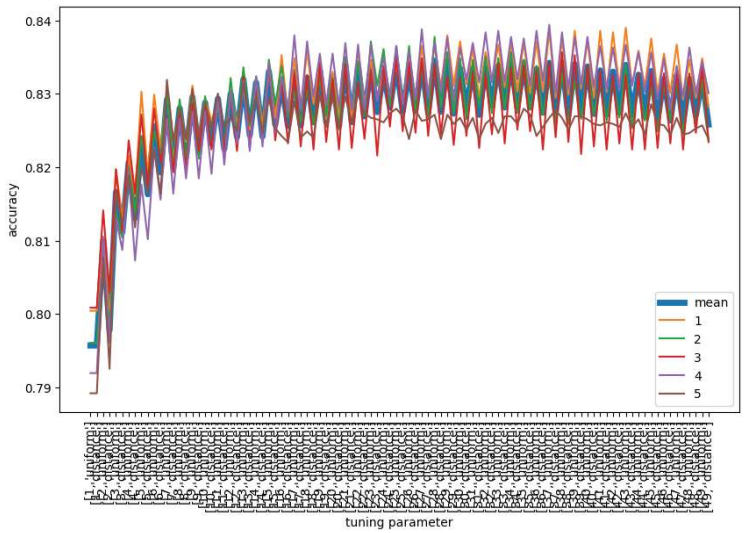

Figure 2: Cross-validation Test Accuracy of Each KNN Model by Folds

Logistic Regression

The hyperparameter C represents the inverse of the regularization strength. A smaller value of C will increase the regularization strength, resulting in simpler models that may be less prone to overfitting. On the other hand, a larger value of C will decrease the regularization strength, resulting in more complex models that may be more prone to overfitting.

Penalty is the regularization penalty to be used. “l1” regularization penalty is the sum of the absolute values of the coefficients. It is used to obtain a sparse model where only the most important features are used in the model. “l2” regularization penalty is the sum of the squares of the coefficients, which is used to prevent overfitting. In this case, “l1” regularization was chosen to promote sparsity of the model.

The solver hyperparameter determines the algorithm used to optimize the logistic regression model. The available options are “newton-cg”, “lbfgs”, “liblinear”, and “sag”. These solvers use different optimization algorithms, and the choice of solver can have an impact on the convergence speed and accuracy of the model.

In the grid search, the optimal hyperparameters were found to be C=1000.0, penalty=“l2”, and solver=“liblinear”. This suggests that the logistic regression model performs better when the regularization strength is relatively strong and L2 regularization is used. Additionally, the “liblinear” solver was found to be optimal, which may indicate that it was the most efficient or effective optimization algorithm for this specific problem.

The feature importance plot shows that `marital-status_ Never-married`, `education-num` and `sex_ Female` are the top three important features contributing to the success of the logistic model. In contrast, there exists some features with negative feature importance, such as `marital-status_ Married-civ-spouse` and `relationship_ Own-child` which means that a random permutation of these features' values may result in a better performance metric.

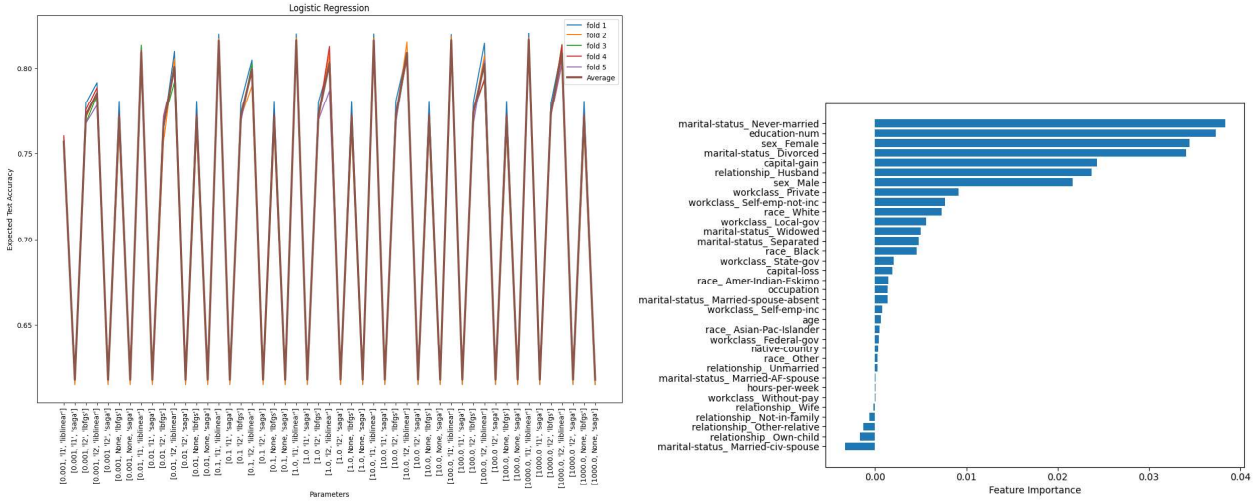

Figure 3: Cross-validation Test Accuracy and Feature Importance of the Best Logistic Mmodel

Support Vector Machine

From the CV plot, we find that C equal to 10 outperforms the others. This is because a larger value of C signifies a higher cost of misclassification, motivating the algorithm to attempt to separate all data points perfectly. The RBF kernel outperforms the other two kernels, which is not surprising as non-parametric models tend to be more complex. The degree of freedom of the polynomial model also supports the idea that a more complex model is a better fit to the data, as a df of 3 performs better than a df of 2. It is hard to explain the reason why sigmoid kernel's accuracy is significantly lower than the other two, since it is a kernel that is hard for humans to interpret(Cortes, et.al). The optimal hyperparameter is C = 10, gamma = auto, kernel = rbf.

We also computed the permutation importance for each feature in this model and identified 9 features with the largest importance values, which are shown in the following plot. It is shown that education_num, capital gain and marital_status_Divorced have significantly higher importance than other features in this SVM model.

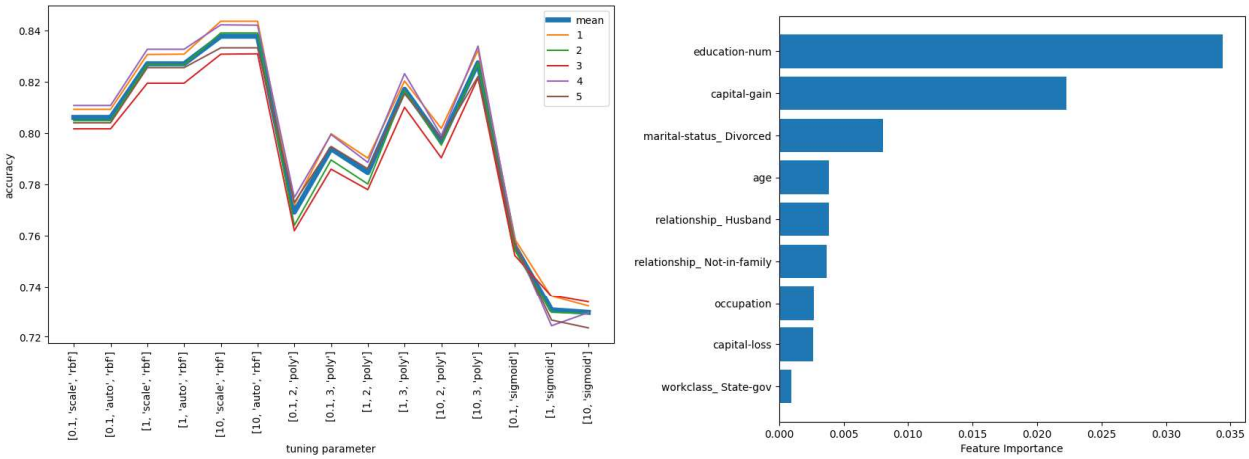

Figure 4: Cross-validation Test Accuracy and Feature Importance of the Best SVM Model

Decision Tree

The CV plot indicates that the accuracy of our models varies widely across different folds. However, some models consistently outperform others, suggesting that certain hyperparameters have a greater impact on the model's performance. Interestingly, we observed that the choice of the criterion hyperparameter, which determines the function used to measure the quality of a split, did not have a significant effect on the results. Both gini and entropy produced similar results in terms of accuracy, suggesting that the choice of criterion may not be crucial for our dataset and model. We have observed that the max_features hyperparameter in our decision tree model has a significant impact on its performance, with the sqrt value leading to lower accuracy. This effect can be explained by the fact that our dataset has a large number of samples compared to the number of features (n » p), and setting a smaller value for max_features can reduce the complexity of the model(Dhiyaussalam et al.), which may result in lower accuracy.

The best model we found is the one with criterion entropy, max_depth 10, max_features none, min_samples_leaf 100, min_samples_split 100. From the permutation importance plot of this best model, we noticed that a large number of features had low permutation importance scores that were close to zero. It's important to note that this doesn't necessarily mean that these features aren't contributing to this model. A possibility is that when some features are highly correlated, permuting one of them would not significantly affect the model's performance, which will lead to low permutation importance score for all of these correlated features(Hooker et al.).

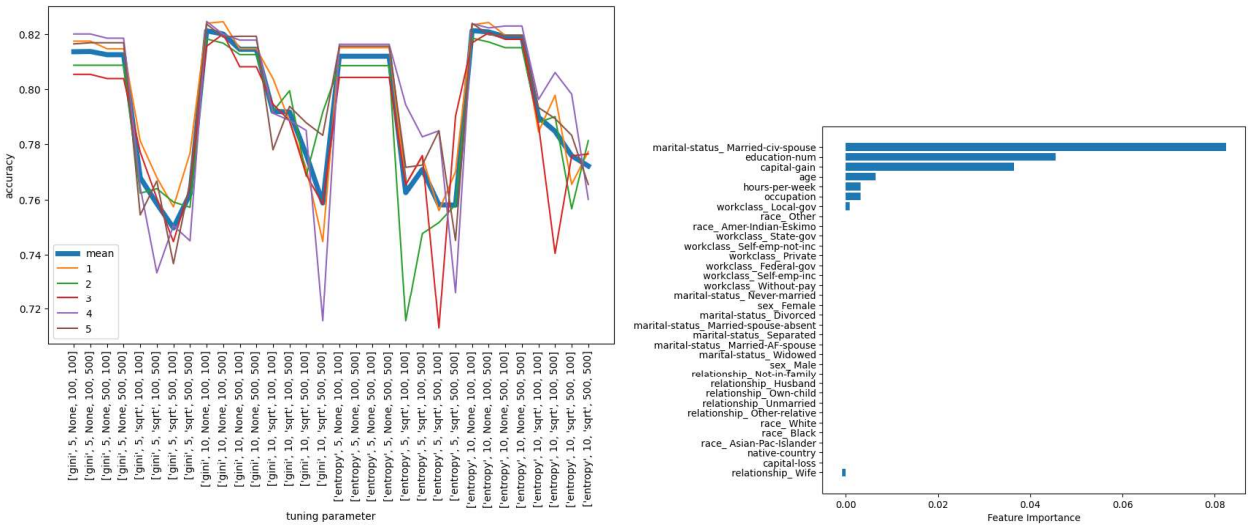

Figure 5: Cross-validation Test Accuracy and Feature Importance of the Best Decision Tree Model

Analysis

We have four metrics to evaluate the models. Misclassification rate is used to evaluate the overall accuracy of a model. Sensitivity, also known as recall, measures how well the model performs when predicting positive cases (e.g., predicting that someone's salary is greater than 50K). Specificity, on the other hand, measures how well the model performs when predicting negative cases (e.g., predicting that someone's salary is less than or equal to 50K). F1 Score is the harmonic mean of precision and recall and ranges between 0 and 1, with 1 being the best possible score.

Table 1: Model Evaluation

Model	Misclassification Rate	Sensitivity	Specificity	F1 Score
KNN	0.16	0.61	0.91	0.65
Logistic Regression	0.21	0.86	0.77	0.67
SVM	0.22	0.85	0.76	0.66
Decision Tree	0.21	0.86	0.77	0.67

Given the imbalanced data, the specificity of the KNN model is much higher compared to its sensitivity. This is because the dataset contains only 25% of data points with a label of 1, making it more likely for the model to perform better with the majority class. For other models, sensitivity and specificity are similar as we balanced the data. K-Nearest Neighbors (KNN) performs best in terms of misclassification rate. However, if the goal is to predict individuals with a salary greater than 50K, logistic regression and decision tree may be better choices because they have higher sensitivity. Thus, the choice of the best model ultimately depends on the specific problem and the evaluation metrics that are most important for that problem.

ROC curves above are very similar for models except SVM, which reflects what we found in the table.

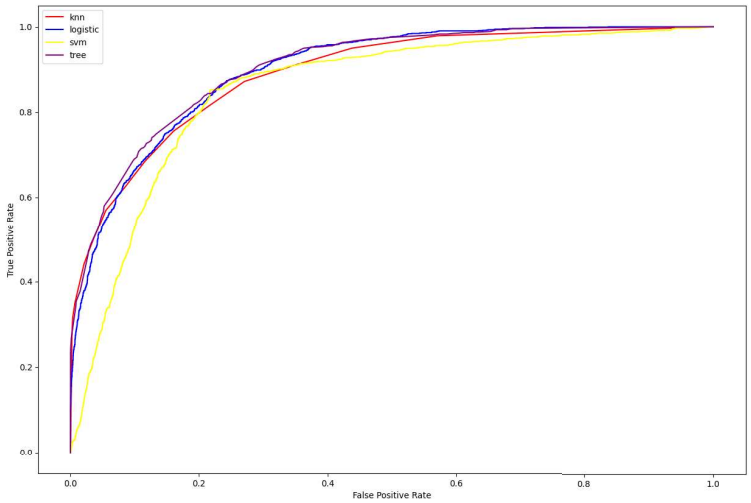

Figure 6: ROC Curves of the 4 Models

To identify the most important variables, we created a table of the top three variables for three models. We do not have most important variables identified for KNN, since KNN model is based on the distance between the data points, and all variables are evaluated equally.

Table 2: Feature Importance

	Logistic Regression	SVM	Decision Tree
Most Important	marital-status_ Never-married	education_num	married-civ-spouse
Second Important	education_num	capital gain	capital gain
Third Important	sex_ Female	marital_status_Divorced	education_num

Our results indicate that marital status and education_num are consistently ranked as the most important variables across all three models. Capital gain also appears twice, suggesting its significance. Additionally, sex could potentially be an important variable as it appears in the logistic regression model. Capital gain represents the profit made from the sale of an asset, such as stocks or real estate. People who have capital gains tend to have more investment income and financial resources, which can be correlated with higher salaries. Education_num is important since certain jobs require specific levels of education, and individuals who have completed the necessary education and training are often qualified for higher paying positions.

Discussion

Our analysis indicates that the models we trained have different performance in various evaluation metrics, and the contribution of each feature to each model's performance varies.

It is important to note that our methodology has some limitations. Since we conducted grid search, we could only consider a limited number of parameters that we deemed reasonable for our predictive model. Due to time and efficiency constraints, we had to select only several options for each hyperparameter. As a result, we cannot guarantee that the model we selected is the optimal one for each method.

More specifically, we have observed that each model has its own strengths and weaknesses. As discussed in the methodology section, we did not apply resampling on the dataset for KNN. Therefore, the model trained by KNN may have a prediction power biased towards the more heavily weighted portion of the data, which

in this case is salary $<$ \$50K. As the results demonstrate, the specificity of this model is significantly higher than its sensitivity.

The logistic regression model assumes a linear relationship between the features and salary, which can be a limitation when the true relationship is more complex.

SVM is particularly effective in dealing with high dimensional data. However, it is computationally expensive and requires a significant amount of time to run. In our case, training the SVM model took over an hour to complete.

Decision trees are easy to interpret and can handle categorical features in an unified way. The model could also produce nonlinear decision boundaries. However, it can be prone to overfitting and may not generalize well to new data, due to the reason that it is very sensitive to even a small number of data points.

Another point to note is that we used permutation importance to evaluate the importance of features in our models. However, this method only highlights the overall importance of each feature to the model's performance, rather than their specific effects on a particular salary category. That being said, we can conclude that marital status, education, age, and capital gain have a significant impact on salary, as these features were found to be highly important in each model we trained.

Moreover, salary is a social variable which can be sensitive to lots of factors. We are only able to consider a limited number of features due to the available data; however, in reality, there are likely many other elements that could affect the salary an individual is earning. A particular example is that there are certain special occupations that earn high compensations in general, whereas in our data, occupation is only categorized into 14 types, which may not be sufficient to explain the variation of real-world job markets.

As discussed above, future research could explore more reasonable options for hyperparameters and consider additional models to train the data. Furthermore, conducting more in-depth analysis on feature importance could provide valuable insights into the factors that have the greatest impact on salaries, which would be beneficial for individuals seeking to improve their earning potential.

References

Ayessa. “Salary Prediction Classification.” Kaggle, 18 Apr. 2022, https://www.kaggle.com/datasets/ayessa/salary-prediction-classification

Cortes, C. and V. Vapnik (1995). Support-vector network. Machine Learning 20, 273– 297

Dhiyaussalam, A. Wibowo, F. A. Nugroho, E. A. Sarwoko and I. M. A. Setiawan, “Classification of Headache Disorder Using Random Forest Algorithm,” 2020 4th International Conference on Informatics and Computational Sciences (ICICoS), Semarang, Indonesia, 2020, pp. 1-5, doi: 10.1109/ICICoS51170.2020.9299105

Hooker, Giles, et al. “Unrestricted Permutation Forces Extrapolation: Variable Importance Requires at Least One More Model, or There Is No Free Variable Importance.” ArXiv.org, 7 Oct. 2021, https://arxiv.org/abs/1905.03151

Kohavi, Ronny, and Barry Becker. UCI Machine Learning Repository: Census Income Data Set, 1 May 1996, https://archive.ics.uci.edu/ml/datasets/Census+Income

Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011

Zheng, A., & Casari, A. (2018). page 97. In Feature Engineering for Machine Learning: Principles and techniques for Data scientists. essay, O’Reilly

Appendix

```
! pip install --upgrade category_encoders

import io
import pandas as pd
import category_encoders as ce
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [15, 10]
import seaborn as sns
from random import choices
from sklearn.model_selection import train_test_split

from sklearn import svm
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV, KFold
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, roc_auc_score, roc_curve
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
import torch
from torch import nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader, random_split
from torchvision.transforms import ToTensor
import torchvision.transforms as transforms
from tqdm import tqdm
import torch.optim as optim
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure

from sklearn.inspection import permutation_importance
%matplotlib inline

from google.colab import files
uploaded = files.upload()

df = pd.read_csv(io.BytesIO(uploaded['salary.csv']))
df.head()

df['salary'].unique()

df.info()

features = df.loc[:, df.columns != 'salary'].columns
classes = df['salary'].unique()

for i in features:
  print(i)
  print(df[i].unique())
```

```
null_workclass = df[df['workclass'] == ' ?']
plt.hist(null_workclass['salary'])
plt.title('salary')
plt.show()
len(null_workclass)/len(df)

# remove rows with ? value
for i in features:
  df = df.loc[df[i] != ' ?']

len(df)

# One Hot Encoding
race_encoder=ce.OneHotEncoder(cols='race',return_df=True,use_cat_names=True)
sex_encoder=ce.OneHotEncoder(cols='sex',return_df=True,use_cat_names=True)
relationship_encoder=ce.OneHotEncoder(cols='relationship',return_df=True,use_cat_names=True)
workclass_encoder=ce.OneHotEncoder(cols='workclass',return_df=True,use_cat_names=True)
marital_status_encoder=ce.OneHotEncoder(cols='marital-status',return_df=True,use_cat_names=True)
df = race_encoder.fit_transform(df)
df = sex_encoder.fit_transform(df)
df = relationship_encoder.fit_transform(df)
df = workclass_encoder.fit_transform(df)
df = marital_status_encoder.fit_transform(df)

# Ordinal Encoding
'''
[{'col': 'occupation',
  'mapping':
   Adm-clerical          1
   Exec-managerial       2
   Handlers-cleaners     3
   Prof-specialty        4
   Other-service         5
   Sales                 6
   Transport-moving      7
   Farming-fishing       8
   Machine-op-inspct     9
   Tech-support         10
   Craft-repair         11
   Protective-serv      12
   Armed-Forces         13
   Priv-house-serv      14
  NaN                   -2
  dtype: int64, 'data_type': dtype('O')}]
'''
occupation_encoder = ce.OrdinalEncoder(cols='occupation', return_df=True)
df = occupation_encoder.fit_transform(df)

'''
[{'col': 'native-country',
  'mapping':
   United-States                 1
   Cuba                          2
```

```
   Jamaica                        3
   India                          4
   Mexico                         5
   Puerto-Rico                    6
   Honduras                       7
   England                        8
   Canada                         9
   Germany                       10
   Iran                          11
   Philippines                   12
   Poland                        13
   Columbia                      14
   Cambodia                      15
   Thailand                      16
   Ecuador                       17
   Laos                          18
   Taiwan                        19
   Haiti                         20
   Portugal                      21
   Dominican-Republic            22
   El-Salvador                   23
   France                        24
   Guatemala                     25
   Italy                         26
   China                         27
   South                         28
   Japan                         29
   Yugoslavia                    30
   Peru                          31
   Outlying-US(Guam-USVI-etc)    32
   Scotland                      33
   Trinadad&Tobago               34
   Greece                        35
   Nicaragua                     36
   Vietnam                       37
   Hong                          38
   Ireland                       39
   Hungary                       40
   Holand-Netherlands            41
  NaN                            -2
  dtype: int64, 'data_type': dtype('O')}]
'''
native_country_encoder = ce.OrdinalEncoder(cols='native-country', return_df=True)
df = native_country_encoder.fit_transform(df)

'''
[{'col': 'salary',
  'mapping':
   <=50K    1
   >50K     2
  NaN      -2
  dtype: int64, 'data_type': dtype('O')}]
'''
```

```
salary_encoder = ce.OrdinalEncoder(cols='salary', return_df=True)
df = salary_encoder.fit_transform(df)

# drop column
df = df.drop(columns=['education', 'fnlwgt'])
df.head()

df['salary'] = df['salary']-1
df['salary'].unique()

df.info()
orig_df = df

df_X = df[df.columns.drop("salary")].to_numpy()
df_y = df["salary"].to_numpy()
# split train and test
train, test = train_test_split(
    df,
    train_size=.8,
    random_state = 441,
    shuffle = True
)

# imbalance train data
plt.hist(train['salary'])
plt.title('salary')
plt.show()

len(train[train['salary'] == 0])/len(train)

# Sampling with Replacement
resampled = train[train['salary'] == 1].sample(frac=3, replace=True, random_state=1)
train = pd.concat([train[train['salary'] == 0], resampled])
plt.hist(train.loc[:, 'salary'])
plt.title('salary')
plt.show()

for i in train.columns:
  print(i)
  print(train[i].unique())

# split features and labels into X and y
X_train = train[train.columns.drop("salary")].to_numpy()
y_train = train['salary'].to_numpy()
X_test = test[test.columns.drop("salary")].to_numpy()
y_test = test['salary'].to_numpy()
features = train[train.columns.drop("salary")].columns.unique()

# for knn
features = orig_df[orig_df.columns.drop("salary")].columns.unique()
orig_df_X= orig_df[orig_df.columns.drop("salary")].to_numpy()
orig_df_Y = orig_df["salary"].to_numpy()
# split data into training and test sets
```

```
orig_X_train, orig_X_test, orig_y_train, orig_y_test = train_test_split(
    orig_df_X,
    orig_df_Y,
    train_size=.8,
    random_state = 441,
    shuffle = True
)

# SVM

model = Pipeline([
    ('scaler', StandardScaler()),
    ('model',  svm.SVC())
])
param_grid = [
  #{'model__C': [0.1, 1, 10], 'model__kernel': ['linear']},
  {'model__C': [0.1, 1, 10], 'model__gamma': ['scale', 'auto'], 'model__kernel': ['rbf']},
    {'model__C': [0.1, 1, 10], 'model__degree': [2, 3], 'model__kernel': ['poly']},
    {'model__C': [0.1, 1, 10], 'model__kernel': ['sigmoid']}
 ]
cv = KFold(
    n_splits=5,  # number of folds
    shuffle=True # protects against data being ordered, e.g., all successes first
)

cv_model = GridSearchCV(
    estimator=model,
    param_grid=param_grid,
    cv=cv
)

cv_model.fit(X=X_train, y=y_train)

mean_sc = cv_model.cv_results_['mean_test_score']

# the higher the score, the less the loss
figure(figsize=(10,6))
x = range(len(mean_sc))
x_axis = [list(d.values()) for d in cv_model.cv_results_['params']]
plt.xticks(x, x_axis,rotation = 90)
plt.plot(x, mean_sc,label = "mean",linewidth = 5)
plt.plot(x, cv_model.cv_results_['split0_test_score'],label = "1")
plt.plot(x, cv_model.cv_results_['split1_test_score'],label = "2")
plt.plot(x, cv_model.cv_results_['split2_test_score'],label = "3")
plt.plot(x, cv_model.cv_results_['split3_test_score'],label = "4")
plt.plot(x, cv_model.cv_results_['split4_test_score'],label = "5")
plt.xlabel('tuning parameter')
plt.ylabel('accuracy')
plt.legend()
plt.show()

cv_model.best_params_
```

```
# test error
1.0 - cv_model.score(X=X_test, y=y_test)

ConfusionMatrixDisplay.from_estimator(
    estimator=cv_model,
    X=X_test,
    y=y_test
)
tn, fp, fn, tp = confusion_matrix(y_test, cv_model.predict(X_test)).ravel()
print('True negative rate(specificity): ', tn / (tn + fp)) #true negative rate(specificity)
print('True positive rate(sensitivity): ', tp / (tp + fn)) #true positive rate(sensitivity)
print('F1 score: ',(2*tp)/(2*tp + fp + fn)) #f1 score
print('Accuracy: ',(tp + tn) / (tn + fp + fn + tp))

perm_importance = permutation_importance(cv_model, X_test, y_test)

feature_names = [train.drop(columns=['salary']).columns]
features = np.array(feature_names)
sorted_idx = perm_importance.importances_mean.argsort()
plt.figure(figsize=(8,7))
plt.barh(features[0][sorted_idx], perm_importance.importances_mean[sorted_idx])
plt.xlabel("Feature Importance")

plt.figure(figsize=(8,7))
plt.barh(features[0][sorted_idx[25:35]], perm_importance.importances_mean[sorted_idx[25:35]])
plt.xlabel("Feature Importance")

# Decision Tree

tree_model = Pipeline([
    ('scaler', StandardScaler()),
    ('model',  tree.DecisionTreeClassifier())
])
tree_param_grid = [
  {'model__criterion': ['gini', 'entropy'], 'model__min_samples_leaf': [100, 500],
   'model__max_depth': [5, 10], 'model__min_samples_split': [100, 500],
   'model__max_features': [None, 'sqrt']}
 ]
cv = KFold(
    n_splits=5,  # number of folds
    shuffle=True # protects against data being ordered, e.g., all successes first
)

tree_cv_model = GridSearchCV(
    estimator=tree_model,
    param_grid=tree_param_grid,
    cv=cv
)

tree_cv_model.fit(X=X_train, y=y_train)

tree_mean_sc = tree_cv_model.cv_results_['mean_test_score']
```

```
figure(figsize=(10,6))
x = range(len(tree_mean_sc))
x_axis = [list(d.values()) for d in tree_cv_model.cv_results_['params']]
plt.xticks(x, x_axis,rotation = 90)
plt.plot(x, tree_mean_sc,label = "mean",linewidth = 5)
plt.plot(x, tree_cv_model.cv_results_['split0_test_score'],label = "1")
plt.plot(x, tree_cv_model.cv_results_['split1_test_score'],label = "2")
plt.plot(x, tree_cv_model.cv_results_['split2_test_score'],label = "3")
plt.plot(x, tree_cv_model.cv_results_['split3_test_score'],label = "4")
plt.plot(x, tree_cv_model.cv_results_['split4_test_score'],label = "5")
plt.xlabel('tuning parameter')
plt.ylabel('accuracy')
plt.legend()
plt.show()

We can see that certain models predict better than the rest.

tree_cv_model.best_params_

# test error
1.0 - tree_cv_model.score(X=X_test, y=y_test)

figure(figsize=(10,6))
tree.plot_tree(tree_cv_model.best_estimator_.named_steps['model'],
               filled=True, feature_names=train.columns)
plt.show()

ConfusionMatrixDisplay.from_estimator(
    estimator=tree_cv_model,
    X=X_test,
    y=y_test
)

tn, fp, fn, tp = confusion_matrix(y_test, tree_cv_model.best_estimator_.predict(X_test)).ravel()
tnr = tn / (tn + fp) #true negative rate(specificity)
tpr = tp / (tp + fn) #true positive rate(sensitivity)
ba = (tpr + tnr) / 2 #balanced accuracy
f1 = (2*tp)/(2*tp + fp + fn) #f1 score
accuracy = (tp + tn) / (tn + fp + fn + tp)

# importance
tree_perm_importance = permutation_importance(tree_cv_model, X_test, y_test)
feature_names = [train.drop(columns=['salary']).columns]
features = np.array(feature_names)
sorted_idx = tree_perm_importance.importances_mean.argsort()
plt.figure(figsize=(8,7))
plt.barh(features[0][sorted_idx], tree_perm_importance.importances_mean[sorted_idx])
plt.xlabel("Feature Importance")

plt.figure(figsize=(8,7))
plt.barh(features[0][sorted_idx[25:35]], tree_perm_importance.importances_mean[sorted_idx[25:35]])
plt.xlabel("Feature Importance")
```

```
# Logistic Regression

param_grid = dict(penalty = ['l1', 'l2', None],
                  C = np.logspace(-3, 3, 7),
                  solver = ['lbfgs', 'liblinear', 'saga'])
print(param_grid)

grid = GridSearchCV(LogisticRegression(), param_grid, cv=5,
                    scoring='accuracy', return_train_score=False)
grid.fit(X_train, y_train)

logistic_result = pd.DataFrame(grid.cv_results_)
logistic_result = logistic_result[-logistic_result['mean_test_score'].isnull()]
logistic_result

labels = list()
for i in list(logistic_result['params'].index):
  labels.append(list(logistic_result['params'][i].values()))
x = range(49)
y_1 = logistic_result['split0_test_score']
y_2 = logistic_result['split1_test_score']
y_3 = logistic_result['split2_test_score']
y_4 = logistic_result['split3_test_score']
y_5 = logistic_result['split4_test_score']
y_avg = logistic_result['mean_test_score']

# plot lines
plt.plot(x, y_1, label = "fold 1")
plt.plot(x, y_2, label = "fold 2")
plt.plot(x, y_3, label = "fold 3")
plt.plot(x, y_4, label = "fold 4")
plt.plot(x, y_5, label = "fold 5")
plt.plot(x, y_avg, label = "Average", linewidth=3)
plt.xticks(x, labels, rotation='vertical')
plt.legend()
plt.xlabel("Parameters")
plt.ylabel("Expected Test Accuracy")
plt.title("Logistic Regression")
plt.show()

print('Best C:', grid.best_estimator_.get_params()['C'])
print('Best Penalty:', grid.best_estimator_.get_params()['penalty'])
print('Best Solver:', grid.best_estimator_.get_params()['solver'])
print('Best mean test error:', grid.best_score_)

best_logistic_model = LogisticRegression(C=1000, penalty='l1', solver='liblinear')
best_logistic_model.fit(X_train, y_train)

logistic_perm_importance = permutation_importance(best_logistic_model, X_test, y_test)
sorted_idx = logistic_perm_importance.importances_mean.argsort()
plt.figure(figsize=(8,7))
plt.barh(features[sorted_idx], logistic_perm_importance.importances_mean[sorted_idx])
plt.xlabel("Feature Importance")
```

```
ConfusionMatrixDisplay.from_estimator(
    estimator=best_logistic_model,
    X=X_test,
    y=y_test
)
tn, fp, fn, tp = confusion_matrix(y_test, best_logistic_model.predict(X_test)).ravel()
print('True negative rate(specificity): ', tn / (tn + fp)) #true negative rate(specificity)
print('True positive rate(sensitivity): ', tp / (tp + fn)) #true positive rate(sensitivity)
print('F1 score: ',(2*tp)/(2*tp + fp + fn)) #f1 score
print('Accuracy: ',(tp + tn) / (tn + fp + fn + tp))

def plot_roc_curve(true_y, y_prob, label, color):
    """
    plots the roc curve based of the probabilities
    """

    fpr, tpr, thresholds = roc_curve(true_y, y_prob)
    plt.plot(fpr, tpr,label = label, color = color)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')

roc_knn = KNeighborsClassifier(n_neighbors=15, weights='uniform')
roc_knn.fit(X=orig_X_train, y=orig_y_train)
y_pred_proba_knn = roc_knn.predict_proba(orig_X_test)[:,1]
roc_svm = SVC(C=10, gamma='auto', kernel='rbf', probability = True)
roc_svm.fit(X_train, y_train)
y_pred_proba_svm = roc_svm.predict_proba(X_test)[:,1]

tree_pipeline = Pipeline([
    ('model', tree.DecisionTreeClassifier(criterion='entropy', max_depth=10,
                                          max_features=None, min_samples_leaf=100,
                                          min_samples_split=100))
])
tree_pipeline.fit(X_train, y_train)
y_pred_proba_tree = tree_pipeline.predict_proba(X_test)[:,1]

plot_roc_curve(orig_y_test, y_pred_proba_knn,"knn","red")
plot_roc_curve(y_test, y_pred_proba,"logistic","blue")
plot_roc_curve(y_test, y_pred_proba_svm,"svm","yellow")
plot_roc_curve(y_test, y_pred_proba_tree,"tree","purple")
plt.legend()
plt.show()


y_pred_proba = best_logistic_model.predict_proba(X_test)[:,1]
plot_roc_curve(y_test, y_pred_proba)
print(f'Logistic Regression AUC score: {roc_auc_score(y_test, best_logistic_model.predict(X_test))}')

# KNN

knn_model = Pipeline([
    ('scaler', StandardScaler()),
    ('model',  KNeighborsClassifier())
```

```
])
knn_param_grid = [
  {'model__n_neighbors': range(1, 50),
  'model__weights': ['uniform', 'distance']}
 ]
cv = KFold(
    n_splits=5,  # number of folds
    shuffle=True # protects against data being ordered, e.g., all successes first
)

knn_cv_model = GridSearchCV(
    estimator=knn_model,
    param_grid=knn_param_grid,
    cv=cv
)


knn_cv_model.fit(X=orig_X_train, y=orig_y_train)

knn_mean_sc = knn_cv_model.cv_results_['mean_test_score']

figure(figsize=(10,6))
x = range(len(knn_mean_sc))
x_axis = [list(d.values()) for d in knn_cv_model.cv_results_['params']]
plt.xticks(x, x_axis,rotation = 90)
plt.plot(x, knn_mean_sc,label = "mean",linewidth = 5)
plt.plot(x, knn_cv_model.cv_results_['split0_test_score'],label = "1")
plt.plot(x, knn_cv_model.cv_results_['split1_test_score'],label = "2")
plt.plot(x, knn_cv_model.cv_results_['split2_test_score'],label = "3")
plt.plot(x, knn_cv_model.cv_results_['split3_test_score'],label = "4")
plt.plot(x, knn_cv_model.cv_results_['split4_test_score'],label = "5")
plt.xlabel('tuning parameter')
plt.ylabel('accuracy')
plt.legend()
plt.show()

knn_cv_model.best_params_

# test error
1.0 - knn_cv_model.score(X=orig_X_test, y=orig_y_test)

ConfusionMatrixDisplay.from_estimator(
    estimator=knn_cv_model,
    X=orig_X_test,
    y=orig_y_test
)
tn, fp, fn, tp = confusion_matrix(orig_y_test, knn_cv_model.predict(orig_X_test)).ravel()
print('True negative rate(specificity): ', tn / (tn + fp)) #true negative rate(specificity)
print('True positive rate(sensitivity): ', tp / (tp + fn)) #true positive rate(sensitivity)
print('F1 score: ',(2*tp)/(2*tp + fp + fn)) #f1 score
print('Accuracy: ',(tp + tn) / (tn + fp + fn + tp))
```