Josef Karpinski
jjk21004
5/6/2025

Exploring Logistic & Linear Regression, SVM, K-NN, and Neural Networks on Real-World Datasets

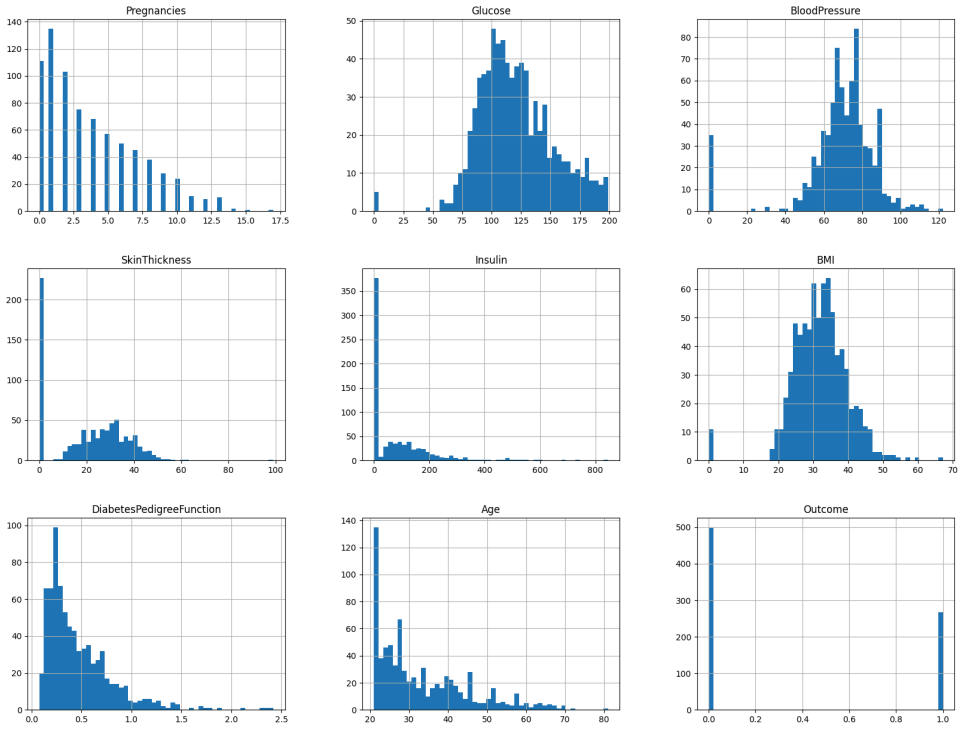
This project explores supervised machine learning models across two real world datasets. The first dataset involves a binary classification task, whether or not a patient has diabetes. The second dataset involves a regression task, predicting median household values in California housing districts. This project discusses machine learning algorithms we learned in class, such as linear regression, logistic regression, SVMs, K-NNs, and neural networks. We will evaluate and compare these modes based on metrics such as accuracy and root mean squared error.

Dataset #1: Diabetes Patients Dataset

For my first dataset, I chose the Diabetes Patient Dataset from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of this dataset is to predict whether or not a patient has diabetes based on given measurements and diagnostics on the patient. I chose this dataset to display different algorithms that we can use in machine learning for binary classification, such as logistic regression, SVM, K-NN, and neural networks. Another reason I chose this dataset is because it features primarily numerical data instead of categorical, making the preprocessing step more straightforward.

Exploring the dataset

Before starting any work on this dataset, it is important to first understand the entire dataset as a whole as well as the context of each of the attributes. This dataset contains the data of 768 patients, 268 of which are diabetic samples, which is roughly 35%. Furthermore, this dataset contains 8 numerical attributes: number of pregnancies, blood glucose level, blood pressure, skin thickness, blood insulin level, BMI, diabetes pedigree function value, and age. Below is a matrix of histograms containing a histogram for each attribute in the dataset.



As we can see, a lot of the attributes have really big spikes for the value of zero, particularly blood glucose level, blood pressure, skin thickness, blood insulin level, and BMI. Obviously, it is not possible for these values to be zero, so this indicates that our dataset has some missing data in these fields. Specifically, it seems like the skin thickness and insulin fields are almost 50% missing data, so we may consider dropping these columns.

Data Preprocessing

It is also important to process and clean our data before running our machine learning algorithms. One critical decision that I made was to remove the insulin and skin thickness attributes from my training data. The other two options were to remove those columns with zeros, which would have removed over half of an already small dataset, or use a preprocessing tool called an imputer, which can transform missing values into the median of the dataset. However, I decided to forego both of these options as they both had their downsides. Simply removing them is realistic, as perhaps these two attributes are hard to obtain in the real world, leading to a more robust model. For the glucose, blood pressure, and BMI columns, I decided to remove any sample that had a zero value, as this did not include a significant amount of samples and to avoid using the imputer. This left myself with 724 samples in the dataset to train and test the models.

Finally, the dataset must be split into training and testing sets, using an 80-20 split. Additionally, for the final step in the preprocessing stage, I used StandardScalar to appropriately scale both the training and test sets based on the training data. This is essential for machine learning models to avoid bias in the attributes.

Methodology and Results

For this classification task, I decided to try out four different models: Logistic Regression, SVM, K-Nearest Neighbors, and neural networks (using MLP). The methodology for training and testing these models was pretty similar, with a few differences in each model's hyperparameters input. In addition to accuracy, I evaluated each model's precision, recall, and F1-score.

For logistic regression, I implemented a model using the default parameters from SciKitLearn with no additional hyperparameters. This model served as a baseline and performed surprisingly well compared to some of the more "sophisticated" models we learned about. Logistic regression was able to achieve an accuracy of 79.3%, as well as an F1-scores of 64.3% and 85.4% for the positive and negative classes respectively.

```
from sklearn.linear_model import LogisticRegression

log_reg_model = LogisticRegression(max_iter=1000)
log_reg_model.fit(X_train_scaled, y_train)
```

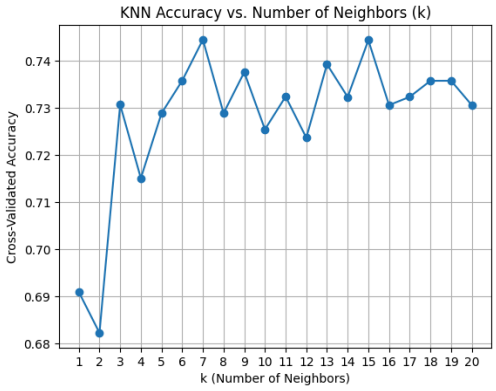
For SVM, I decided to go a little more in-depth than the logistic regression model since we learned about hyperparameter tuning in relation to SVM in lecture. I used GridSearchCV, which searches hyperparameter combinations with cross validation, to find the optimal SVM model for this task. The best model found used a linear kernel and a C value of 0.1. When testing, this SVM was able to achieve an accuracy of 80.0%, as well as an F1-scores of 65.1% and 86.0% for the positive and negative classes respectively. This was the best performing model that I found with this dataset, most likely because of the hyperparameter tuning that took place.

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

param_grid = [
{
    'kernel': ['linear', 'rbf'],
    'gamma': [0.1,1,2,5,10,30,100],
    'C': [0.1,1,2,5,10,30,100]
```

```
}]

svm_clf = SVC()
grid_search = GridSearchCV(svm_clf, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)
```

For K-Nearest Neighbors, I tested 20 different k values (from 1-20) using cross validation to find the optimal value of k-neighbors to use for a final model. From evaluating the graph below, I decided to choose k=7 as it had one of the highest validation accuracies out of the ones tested, as well as being the point where the validation accuracy improvement starts to taper off. When evaluating this k=7 K-NN on the test dataset, this was able to achieve an accuracy of 78.6%, as well as an F1-scores of 66.0% and 84.4% for the positive and negative classes respectively.

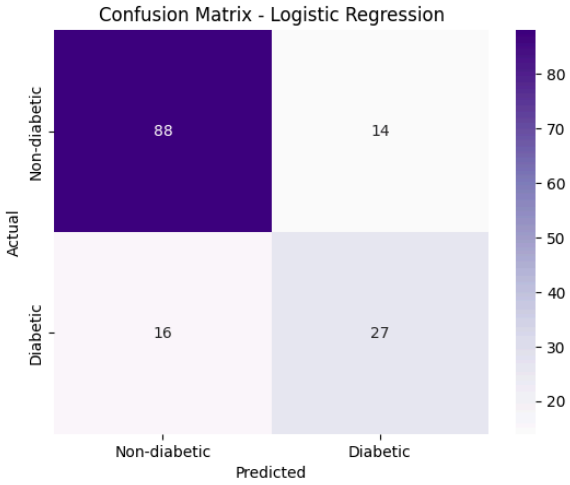
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
k_values = range(1, 21)
accuracies = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train_scaled, y_train, cv=5, scoring='accuracy')
    accuracies.append(scores.mean())
```

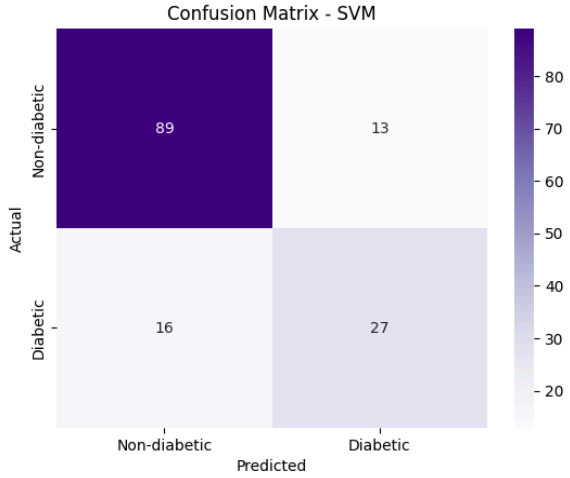


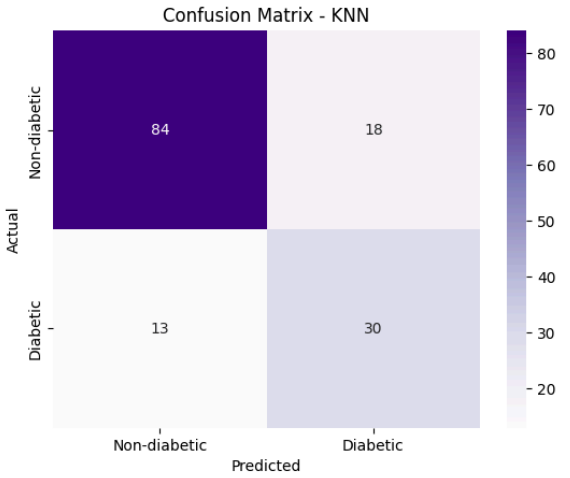

```
knn_final = KNeighborsClassifier(n_neighbors=7)
knn_final.fit(X_train_scaled, y_train)
```

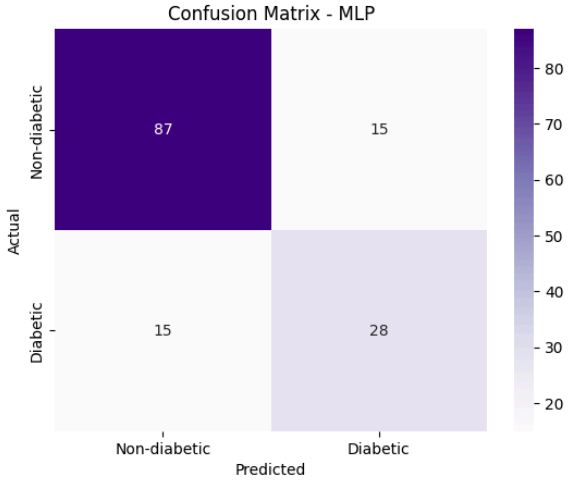
For neural networks, I used an MLPClassifier to implement a simple neural network. We didn't cover this material too much in lecture, but I wanted to include it just to see how it would perform on this type of dataset without too much tuning and tweaking. This model was able to achieve an accuracy of 79.3%, as well as an F1-scores of 65.1% and 85.3% for the positive and negative classes respectively.

```
from sklearn.neural_network import MLPClassifier
mlp_clf = MLPClassifier(hidden_layer_sizes=(200,), max_iter=500, random_state=42)
mlp_clf.fit(X_train_scaled, y_train)
```







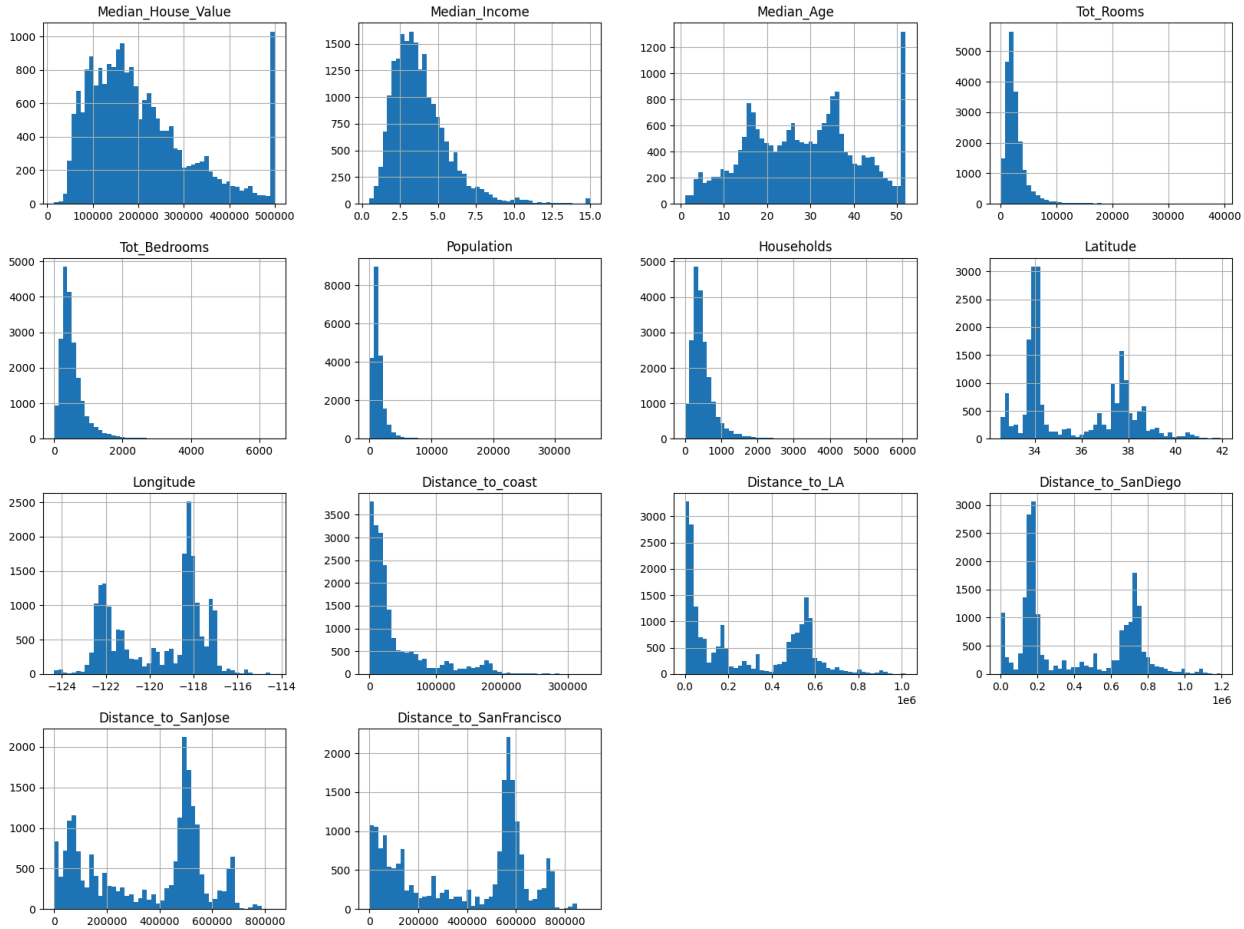


Dataset #2: California Housing Prices

For the second dataset, I chose the California Housing Prices Dataset, which was formed using 1990 California Census data. This dataset contains data on California housing districts and their statistics. The main objective of this dataset is to predict the median house value in each district. For this task, I tried multiple different machine learning regression algorithms that we learned about in this course, such as linear regression, SVM, K-NN, and neural networks.

Exploring the dataset

This dataset contains data for around 21000 California housing districts. Each sample contains the median house value, and then also 13 numerical attributes on the housing district, such as median income, median age, total bedrooms, location, proximity to different cities, and more. Below is a matrix of histograms for each attribute included in the dataset.



As for the need for data cleaning, it doesn't seem like this dataset has too much messy data. For example, there are no missing fields like we had in the diabetes dataset. Two attributes, the median house value and median age, seem to have some sort of maximum value as seen by the massive spikes at the end of their histograms. I wasn't sure exactly how to handle this, perhaps I could have removed these sorts of values, but then that would have disregarded quite a bit of the dataset, so I decided to leave these samples untouched.

I decided to engineer 3 more features into the dataset. As you can see, the dataset contains statistics such as number of households, number of rooms, number of bedrooms, and population. The problem with these attributes is that some districts will have more households than others, so the latter 3 attributes may not be indicative of the true values of the homes in that district. To solve this, I created 3 new features, rooms per household, bedrooms per household, and people per household.

```
df['rooms_per_household'] = df['Tot_Rooms']/df['Households']
df['bedrooms_per_household'] = df['Tot_Bedrooms']/df['Households']
df['population_per_household'] = df['Population']/df['Households']
```

Methodology

For this regression task, I decided to try out four different models: Linear Regression, SVM, K-Nearest Neighbors, and neural networks (using MLP). The methodology for training and testing these models was pretty similar, with a few differences in each model's hyperparameters input. For evaluating the results and predictions of each model, I decided to use root mean squared error (RMSE).

For a simple baseline model, I decided to use linear regression. This was a good choice to start off as linear regression is pretty low effort to code and train, so it will give us a good benchmark for our other algorithms. This model was able to achieve an RMSE of 69888. For reference, the mean target value was around 200000. I also tried Lasso regression and Ridge regression, however, these did not offer significant improvement on the model.

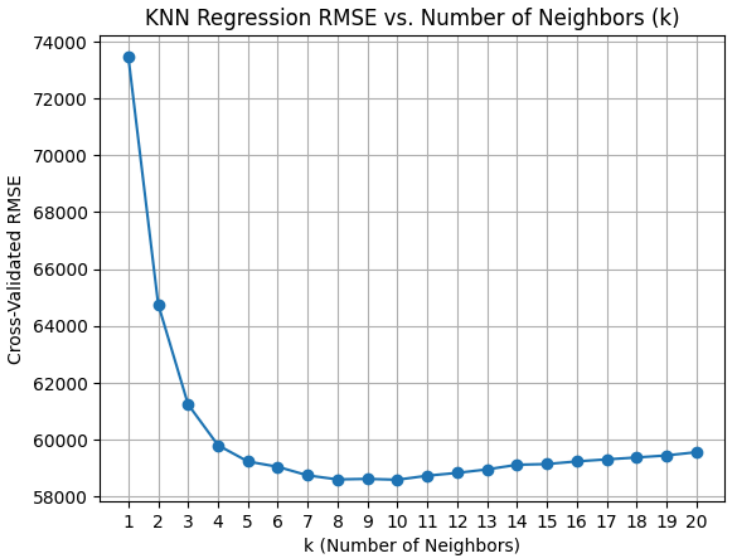
```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(X_train_scaled, y_train)
```

For SVM, I decided to once again go a little more in-depth and try multiple hyperparameter values in the regressor. Since this dataset is so large, I wasn't able to try as many

kernel and hyperparameter values because of training times, but I still tried various C values for a linear kernel. The best model found using cross validation had a C value of 30, and had a RMSE of 74949. This surprisingly did worse than the linear regression model.

For K-NN, I once again decided to run the algorithm for many different values of k (1-20) and analyze them using cross validation. As we can see from the graph below, the best performing k value was k=8, any higher values produced worse performance. When running k=8 K-NN on the test set, it achieved a RMSE of 58070, a considerable improvement from the two previous models.

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import cross_val_score

k_values = range(1, 21)
rmses = []

for k in k_values:
    knn = KNeighborsRegressor(n_neighbors=k)
    neg_mse_scores = cross_val_score(knn, X_train_scaled, y_train, cv=5,
scoring='neg_mean_squared_error')
    rmse_scores = [np.sqrt(-mse) for mse in neg_mse_scores]
    rmses.append(sum(rmse_scores) / len(rmse_scores))
```




```
knn_final = KNeighborsRegressor(n_neighbors=8)
knn_final.fit(X_train_scaled, y_train)
```

For neural networks, I used MLPRegressor for the neural network. I tried various different network sizes, and settled on a two layer network with 100 neurons in the first layer and

50 in the second. This achieved a RMSE of 56279 and was the best performing model out of the four I tried.

```
from sklearn.neural_network import MLPRegressor

mlp_reg = MLPRegressor(
    hidden_layer_sizes=(100, 50),
    max_iter=1000,
    early_stopping=True,
    learning_rate_init=0.001,
    random_state=42
)
mlp_reg.fit(X_train_scaled, y_train)
```

Comparing the Results

Comparing the results from each dataset's experiment, we can see some interesting trends. Obviously, it's impossible to compare the actual metrics from each experiment, as the diabetes dataset was classification, and the housing data was regression.

It was interesting to see how the more complex models compared to more simple models. For example, in the classification task, the MLP classifier was not able to beat logistic regression in terms of accuracy, but in the regression task, the MLP regressor was able to significantly improve on linear regression. I think a main cause for this would be the size of the datasets. The diabetes dataset was pretty small and contained very few complete attributes, so perhaps this made it so the MLP classifier was not able to show its ability to capture more complex behavior in the dataset. On the other hand, the California housing data had over 21000 samples and a lot more features, allowing the neural network to perform better.

Another interesting outcome was the results of the K-NN models. Initially, I thought that since the algorithm for K-NN is so simple that it would not be able to outperform more complex models like SVM and MLP. However, this was not at all the case, K-NN was not the definitive best model for either of the tasks, but it came very close for both, showing promising results. In the housing predictor specifically, it offered very impressive results, improving on linear regression and SVM by 15-20% in terms of RMSE.

Conclusion

In this project I was able to apply both classification and regression machine learning models to the diabetes and California housing datasets. This included models such as linear regression, logistic regression, SVM classifier/regressor, K-NN classifier/regressor, and MLP classifier/regressor (neural networks). For future improvement upon these results, I would seek more broad and generalizable datasets, as well as improving prediction accuracy with fine-tuning and perhaps trying deep learning models for these tasks.

References

Diabetes Dataset:
https://www.kaggle.com/datasets/akshaydattatraykhare/diabetes-dataset

California Housing Prices Dataset:
https://www.kaggle.com/datasets/fedesoriano/california-housing-prices-data-extra-features