MET CS677 Data Science with Python
Assignment 3

Alan Szmyt

April 5, 2023

In this assignment, we will implement k-NN and logistic regression classifiers to detect "fake" banknotes and analyze the comparative importance of features in predicting accuracy.

For the dataset, we use "banknote authentication dataset" from the machine learning depository at UCI:

https://archive.ics.uci.edu/ml/datasets/banknote+authentication.

Dataset Description: From the website: "This dataset contains 1,372 examples of both fake and real banknotes. Data were extracted from images that were taken from genuine and forged banknote-like specimens. For digitization, an industrial camera usually used for print inspection was used. The final images have 400 x 400 pixels. Due to the object lens and distance to the investigated object gray-scale pictures with a resolution of about 660 dpi were gained. Wavelet Transform tool were used to extract features from images."

1. f_1 - variance of a wavelet transformed image.
2. f_2 - skewness of a wavelet transformed image.
3. f_3 - kurtosis of a wavelet transformed image.
4. f_4 - entropy of image.
5. class (integer)

In other words, assume that you have a machine that examines a banknote and computes 4 attributes (step 1). Then each banknote is examined by a much more expensive machine and/or by human expert(s) and classified as fake or real (step 2). The second step is very time-consuming and expensive. You want to build a classifier that would give your results after step 1 only.

We assume that class 0 are good banknotes. We will use color **"green"** or " $+$ " for legitimate banknotes. Class 1 are assumed to be fake banknotes, and we will use color **"red"** or " $-$ " for counterfeit banknotes. These are the "true" labels.

Cell 02

```
# Python dependency imports.
import operator
import pandas as pd
import numpy as np
import seaborn as sns
from IPython.display import Latex, SVG
from pandas import DataFrame, Series
from pathlib import Path
import matplotlib.pyplot as plt
from collections import defaultdict
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    confusion_matrix,
    accuracy_score,
    ConfusionMatrixDisplay
)
from assignment3 import (
    BankNote,
    bu_id_df,
    feature_stats_table,
    features_to_plot,
    knn_accuracy,
    log_reg_accuracy,
)
from utils import (
    create_latex_table,
    dataframe_memory_usage,
    artifacts,
    relative_path_to,
    rename_columns,
    resources,
    Color,
    save_figures,
    get_truth_statistics,
    get_truth_statistics_df,
    TruthStats,
)
from constants import (
    COL_CLASS,
    COL_COLOR,
    COL_F1,
    COL_F2,
    COL_F3,
    COL_F4,
    COL_PREDICTION,
    GOOD_BILLS,
    BANKNOTES,
    GOOD_BILLS_GREEN,
    FAKE_BILLS,
    FAKE_BILLS_RED,
    INITIAL_COLS,
    features_to_latex,
    stats_to_latex,
)
```

Cell 03

```
# Global Seaborn options.
sns.set_theme(font_scale=1.5, rc={"text.usetex": True})

# Global pandas options.
pd.set_option("display.max_rows", 10)
pd.set_option("display.max_colwidth", None)
pd.set_option("display.max_seq_items", 50)
pd.set_option("display.show_dimensions", False)
pd.set_option("display.expand_frame_repr", False)
pd.set_option("mode.chained_assignment", "raise")
pd.set_option("display.precision", 4)
pd.set_option("styler.format.precision", 4)

cwd: Path = Path.cwd()

# Banknote dataset file from UCI.
banknote_dataset_file: Path = \
    resources.joinpath("data_banknote_authentication.txt")

print(
    f"Loading dataset from: "
    f"{relative_path_to(cwd, banknote_dataset_file)}"
)
```

```
Loading dataset from: resources/data_banknote_authentication.txt
```

1 Question 1

1.1 Load the data into a dataframe and add column “color“. For each class 0, this should contain “green“ and for each class 1 it should contain “red“.

Answer:

Cell 04

```
# dtype mapping to use for the csv file. Set class to categorical.
dtypes = defaultdict(np.float64, {COL_CLASS: pd.CategoricalDtype.name})

# Loading the data_banknote_authentication.txt file into a dataframe.
banknote_dataset: DataFrame = pd.read_csv(
    banknote_dataset_file,
    header=None,
    names=INITIAL_COLS,
    dtype=dtypes
)

# Add color column to dataframe based upon the class column.
banknote_dataset[COL_COLOR] = banknote_dataset.apply(
    lambda row: Color.GREEN.lower
    if int(row[COL_CLASS]) == BankNote.LEGITIMATE
    else Color.RED.lower,
    axis=1
).astype("category")

# Build latex table to render in the document.
banknote_table = create_latex_table(
    rename_columns(banknote_dataset, features_to_latex).head(),
    label="tab:question1_1",
    caption="Question 1.1"
)
Latex(banknote_table)
```

Table 1: Question 1.1

	f_1	f_2	f_3	f_4	class	color
0	3.6216	8.6661	-2.8073	-0.4470	0	green
1	4.5459	8.1674	-2.4586	-1.4621	0	green
2	3.8660	-2.6383	1.9242	0.1065	0	green
3	3.4566	9.5228	-4.0112	-3.5944	0	green
4	0.3292	-4.4552	4.5718	-0.9888	0	green

1.2 For each class and for each fixture f_1, f_2, f_3, f_4, compute its mean $\mu()$ and standard deviation $\sigma()$. Round the results to 2 decimal places and summarize them in a table as show below:

Answer:

Cell 05

```
# Compute the feature distribution stats table.
feature_stats: DataFrame = feature_stats_table(
    banknote_dataset.drop(columns=COL_COLOR)
)

# Build latex table to render in the document.
banknote_stats_table = create_latex_table(
    rename_columns(feature_stats, stats_to_latex).head(),
    label="tab:question1_2",
    caption="Question 1.2"
)
Latex(banknote_stats_table)
```

Table 2: Question 1.2

	$\mu(f_1)$	$\sigma(f_1)$	$\mu(f_2)$	$\sigma(f_2)$	$\mu(f_3)$	$\sigma(f_3)$	$\mu(f_4)$	$\sigma(f_4)$
0	2.2800	2.0200	4.2600	5.1400	0.8000	3.2400	-1.1500	2.1300
1	-1.8700	1.8800	-0.9900	5.4000	2.1500	5.2600	-1.2500	2.0700
all	0.4300	2.8400	1.9200	5.8700	1.4000	4.3100	-1.1900	2.1000

1.3 Examine your table. Are there any obvious patterns in the distribution of banknote in each class.

Answer:

The most notable pattern that I observed right away is the difference between the mean values for f_1 (variance) and f_2 (skewness) and see that the standard deviation values for both are hight aswell. f_2 also stands out because it has the highest standard deviation. Also for the f_4 (entropy). The values are very close for both good and fake banknotes, so that may be hard to know the difference based upon entropy alone.

2 Question 2

2.1 Split your dataset X into training X_{train} and X_{test} parts (50/50 split). Using "pairplot" from the seaborn package, plot pairwise relationships in X_{train} separately for class 0 and class 1. Save your results into 2 pdf files ``good_bills.pdf`` and ``fake_bills.pdf``

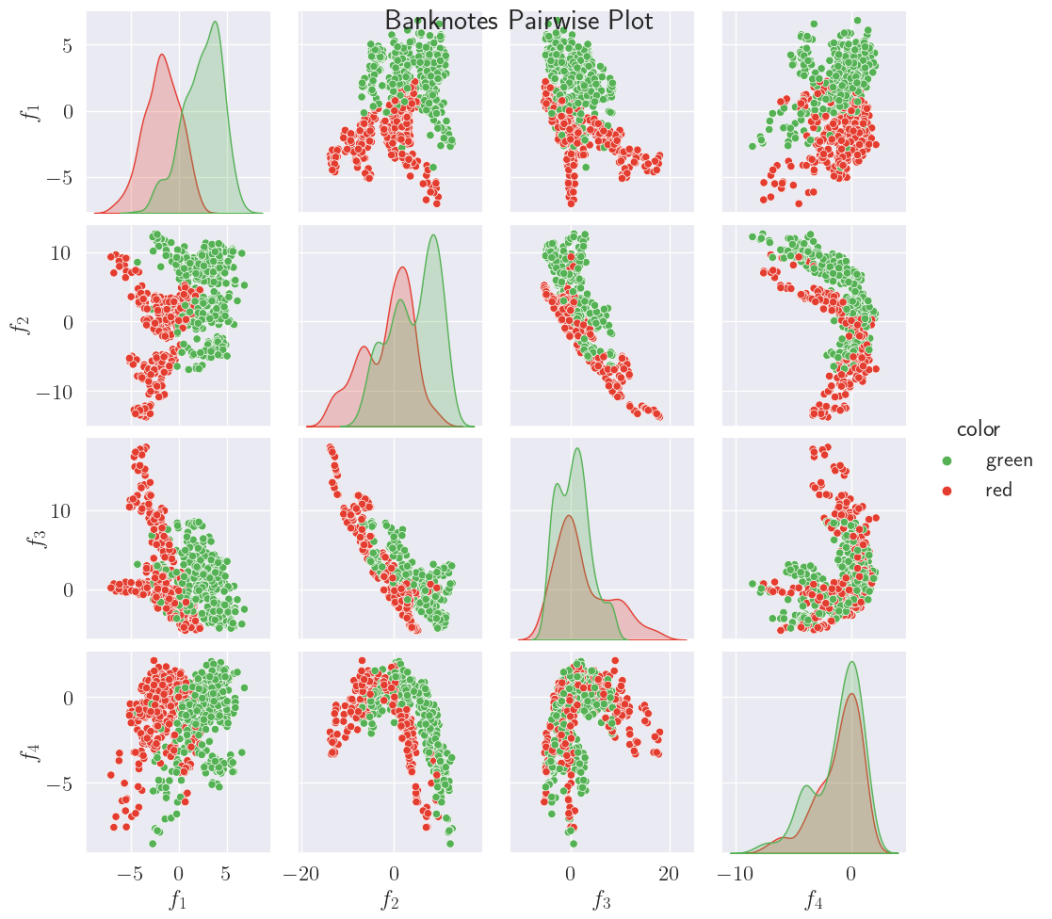
Answer:

Cell 06

```
# Split the dataset into training/testing datasets split 50/50.
y: Series = banknote_dataset[COL_CLASS]
x_train, x_test, y_train, y_test = train_test_split(
    banknote_dataset,
    y,
    test_size=0.5, random_state=42, stratify=y
)

# Training subset of rows with legitimate banknotes.
train_good_bills: DataFrame = x_train.loc[
    x_train[COL_CLASS].astype(int) == BankNote.LEGITIMATE
]

# Training subset of rows with counterfeit banknotes.
train_bad_bills: DataFrame = x_train.loc[
    x_train[COL_CLASS].astype(int) == BankNote.COUNTERFEIT
]
```

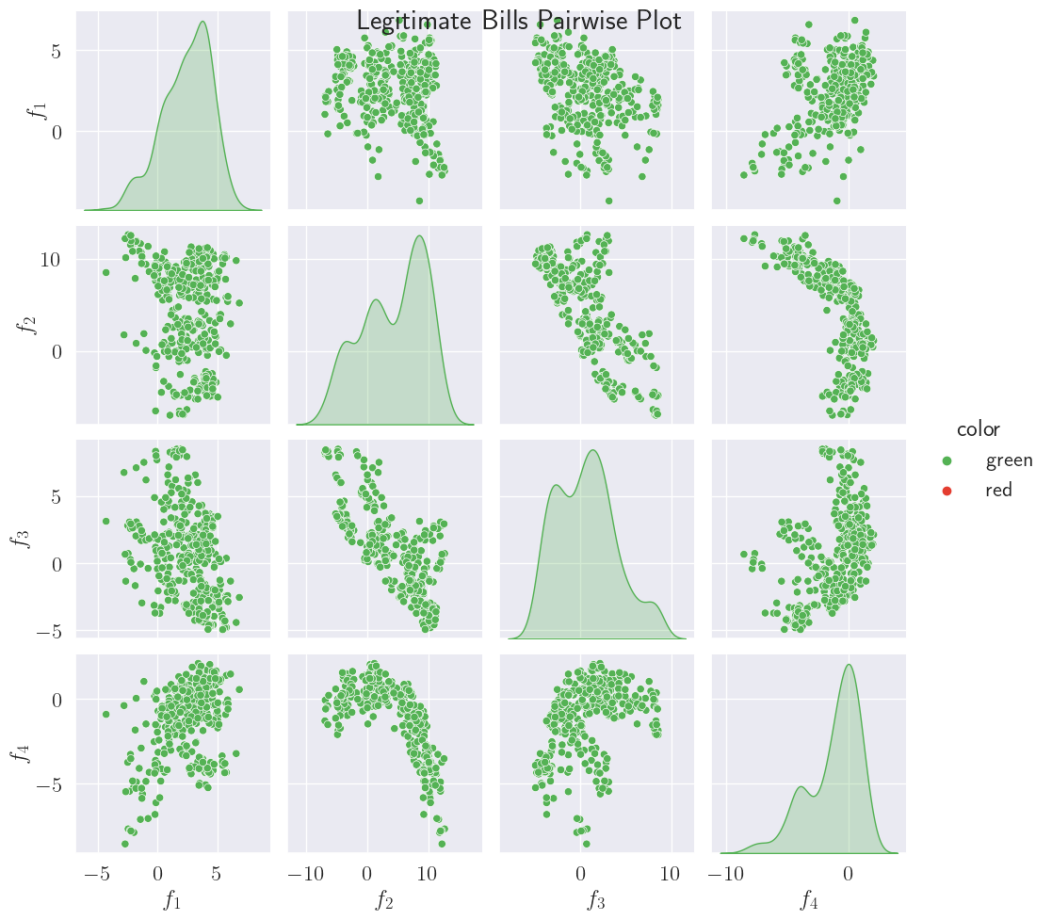
Cell 07

```
# Create, save, and plot a pairwise plot of the all bills.
train_pairwise = sns.pairplot(
    rename_columns(x_train, features_to_plot),
    hue="color",
    palette=[GOOD_BILLS_GREEN, FAKE_BILLS_RED],
)
train_pairwise.fig.suptitle("Banknotes Pairwise Plot")
plt.show()
```



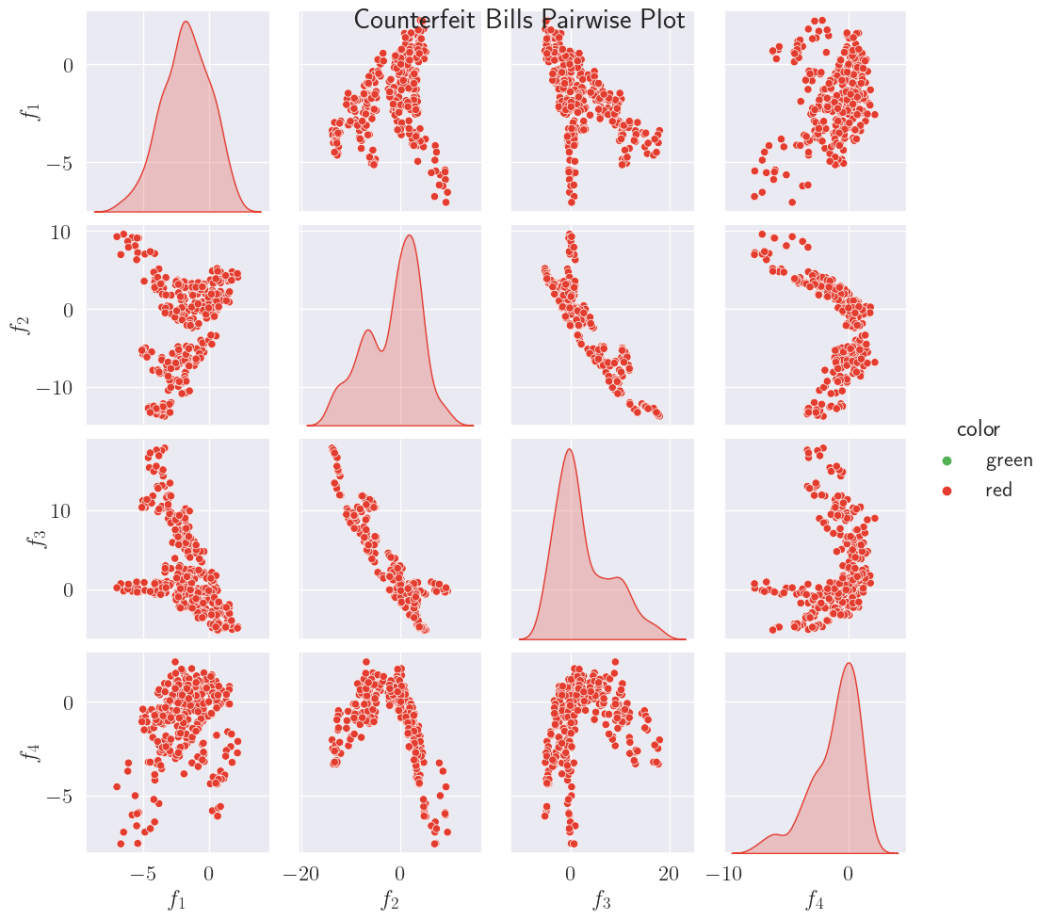
Cell 08

```
# Create, save, and plot a pairwise plot of the good bills.
good_pairwise = sns.pairplot(
    rename_columns(train_good_bills, features_to_plot),
    hue="color",
    palette=[GOOD_BILLS_GREEN, FAKE_BILLS_RED]
)
good_pairwise.fig.suptitle("Legitimate Bills Pairwise Plot")
plt.show()
```



Cell 09

```
# Create, save, and plot a pairwise plot of the fake bills.
fake_pairwise = sns.pairplot(
    rename_columns(train_bad_bills, features_to_plot),
    hue="color",
    palette=[GOOD_BILLS_GREEN, FAKE_BILLS_RED]
)
fake_pairwise.fig.suptitle("Counterfeit Bills Pairwise Plot")
plt.show()
```



2.2 Visually examine your results. Come up with three simple comparisons that you think may be sufficient to detect a fake bill. For example, your classifier may look like this:

```
# assume you are examining a bill
# with features f_1, f_2, f_3, and f_4
# your rule may look like this:
if (f_1 > 4) and (f_2 > 8) and (f_4 < 25):
    x = "good"
else:
    x = "fake"
```

Answer:

Cell 10

```
def simple_classifier(row: Series) -> int:
    """Simple classifier to predict if banknotes are counterfeit or legitimate.

    Args:
        row (Series): The row of the dataframe to predict the class label for.

    Returns:
        int: 0 if the label is legitimate, 1 if the label is fake.
    """
    if row[COL_F1] < 4 and row[COL_F2] < 10 < row[COL_F3]:
        return BankNote.COUNTERFEIT.value
    else:
        return BankNote.LEGITIMATE.value
```

2.3 Apply your simple classifier to X_{test} and compute predicted class labels.

Answer:

Cell 11

```
# Apply the simple classifier to predict class labels.
x_test[COL_PREDICTION] = x_test.apply(simple_classifier, axis=1)
```

2.4 Comparing your predicted class labels with true labels, compute the following:

(a) TP - true positives (your predicted label is " + " and true label is " + ")

(b) FP - false positives (your predicted label is " + " but true label is " − ")

(c) TN - true negatives (your predicted label is " − " and true label is " − ")

(d) FN - false negatives (your predicted label is " − " but true label is " + ")

(e) TPR = TP/(TP + FN) - true positive rate. This is the fraction of positive labels that you predicted correctly. This is also called sensitivity, recall or hit rate.

(f) TNR = TN/(TN + FP) - true negative rate. This is the fraction of negative labels that you predicted correctly. This is also called specificity or selectivity.

Answer:

Cell 12

```
# Compute the confusion matrix statistics for the simple classifier.
truth_statistics: DataFrame = get_truth_statistics(
    x_test, COL_CLASS, COL_PREDICTION
)

true_positives: int = \
    truth_statistics[TruthStats.TRUE_POSITIVE.value].iloc[0]
print(f"True positives: {true_positives}")

false_positives: int = \
    truth_statistics[TruthStats.FALSE_POSITIVE.value].iloc[0]
print(f"False positives: {false_positives}")

true_negatives: int = \
    truth_statistics[TruthStats.TRUE_NEGATIVE.value].iloc[0]
print(f"True negatives: {true_negatives}")

false_negatives: int = \
    truth_statistics[TruthStats.FALSE_NEGATIVE.value].iloc[0]
print(f"False negatives: {false_negatives}")

true_positive_rate: int = \
    truth_statistics[TruthStats.TRUE_POSITIVE_RATE.value].iloc[0]
print(f"True positive rate: {true_positive_rate}")

true_negative_rate: int = \
    truth_statistics[TruthStats.TRUE_NEGATIVE_RATE.value].iloc[0]
print(f"True negative rate: {true_negative_rate}")
```

```
True positives: 381
False positives: 278
True negatives: 27
False negatives: 0
True positive rate: 1.0
True negative rate: 0.08852459016393442
```

2.5 Summarize your findings in the table as shown below:

Answer:

Cell 13

```
# Build latex truth statistics table to render in the document.
truth_statistics_table = create_latex_table(
    truth_statistics,
    label="tab:question2_5",
    caption="Question 2.5"
)
Latex(truth_statistics_table)
```

Table 3: Question 2.5

	TP	FP	TN	FN	accuracy	TPR	TNR
0	381	278	27	0	0.5900	1.0000	0.0885

2.6 Does your simple classifier give you higher accuracy on identifying "fake" bills or "real" bills. Is your accuracy better than 50% ("coin" flipping)?

Answer:

Cell 14

```
# Is the simple classifier's accuracy is better than a coin flip?
simple_accuracy: float = \
    truth_statistics.iloc[0][TruthStats.ACCURACY.value] * 100
print(f"Simple classifier's accuracy: {simple_accuracy}")
if simple_accuracy >= 50:
    print("Accuracy for simple classifier is better than a coin flip!")
else:
    print("Accuracy for simple classifier is worse than a coin flip!")
```

```
Simple classifier's accuracy: 59.0
Accuracy for simple classifier is better than a coin flip!
```

3 Question 3: (use k-NN classifier using sklearn library)

3.1 Take $k = 3, 5, 7, 9, 11$. Use the same X_{train} and X_{test} as before. For each k, train your k-NN classifier on X_{train} and compute its accuracy for X_{test}.

Answer:

Cell 15

```
# n_neighbor values to try with the kNN classifier.
k_vals: list[int] = list(range(3, 12, 2))

# Split the dataset into training/testing datasets split 50/50.
y: Series = banknote_dataset[COL_CLASS]
x_train, x_test, y_train, y_test = train_test_split(
    banknote_dataset.drop([COL_COLOR, COL_CLASS], axis=1),
    y,
    test_size=0.5, random_state=42, stratify=y
)

# Feature scaling, generally necessary for kNN.
scaler: StandardScaler = StandardScaler()
x_train_sc: np.ndarray = scaler.fit_transform(x_train)
x_test_sc: np.ndarray = scaler.transform(x_test)

# Save the accuracies for each n_neighbor value.
knn_accuracies: dict[int, float] = {}
knn_error: dict[int, float] = {}
for k_val in k_vals:
    # k-Nearest Neighbor classifier for classifying banknotes.
    classifier: KNeighborsClassifier = KNeighborsClassifier(
        n_neighbors=k_val, p=2, metric="euclidean"
    )
    classifier.fit(x_train_sc, y_train)

    # Use classifier to make predictions.
    y_pred: np.ndarray = classifier.predict(x_test_sc)

    # Compute the accuracy for k and save.
    knn_accuracies.update({k_val: accuracy_score(y_test, y_pred)})

    # Compute the error rate of k and save.
    # Reference: https://stackoverflow.com/a/62616556
    knn_error.update({k_val: np.mean(y_pred != y_test)})

for k_val, knn_acc in knn_accuracies.items():
    print(f"Prediction accuracy for k={k_val}: {knn_acc}")
```

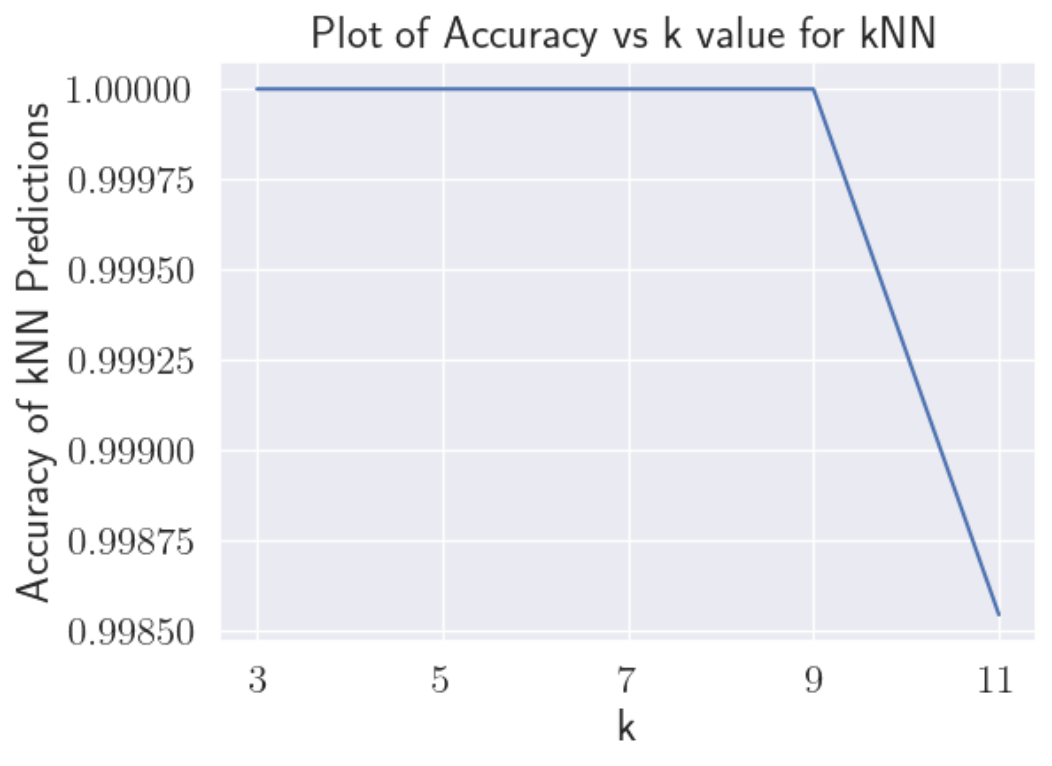
```
Prediction accuracy for k=3: 1.0
Prediction accuracy for k=5: 1.0
Prediction accuracy for k=7: 1.0
Prediction accuracy for k=9: 1.0
Prediction accuracy for k=11: 0.9985422740524781
```

3.2 Plot a graph showing the accuracy. On x axis you plot k and on y-axis you plot accuracy. What is the optimal value k^* of k?

Answer:

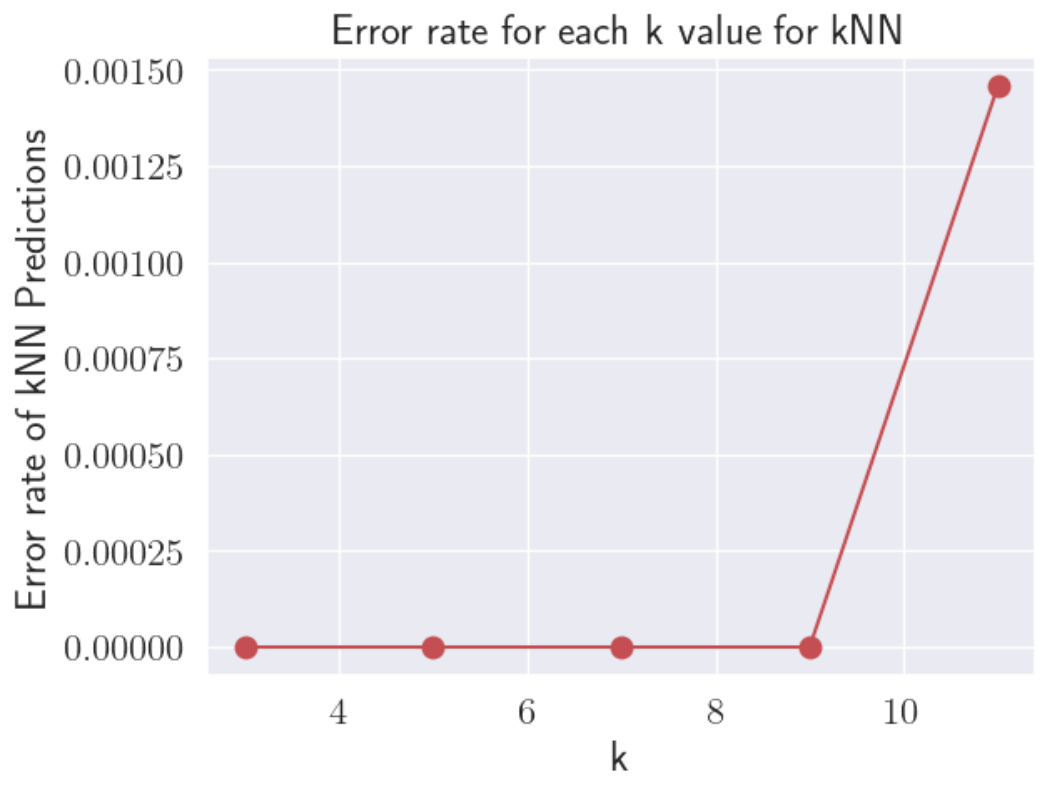
Cell 16

```
# Plot a graph showing the accuracies of each k.
plt.plot(k_vals, list(knn_accuracies.values()))
plt.xticks(k_vals)
plt.xlabel("k")
plt.ylabel("Accuracy of kNN Predictions")
plt.title("Plot of Accuracy vs k value for kNN")
plt.gcf().subplots_adjust(bottom=0.2, left=0.18)
plt.show()
```



Cell 17

```
# Error rate plot.
plt.plot(k_vals, knn_error.values(), color="r", marker='o', markersize=9)
plt.xlabel("k")
plt.ylabel("Error rate of kNN Predictions")
plt.title("Error rate for each k value for kNN")
plt.show()
```



The optimal value k^* should be the highest k that has the least amount of error. Because this dataset is well formatted, the k-NN classifier was successful for all k values, so I will pick $k = 9$ as my k^*. Some additional readings point to that choosing an optimal k value can be done using cross-validation techniques.

Reference:

https://neptune.ai/blog/cross-validation-in-machine-learning-how-to-do-it-right.

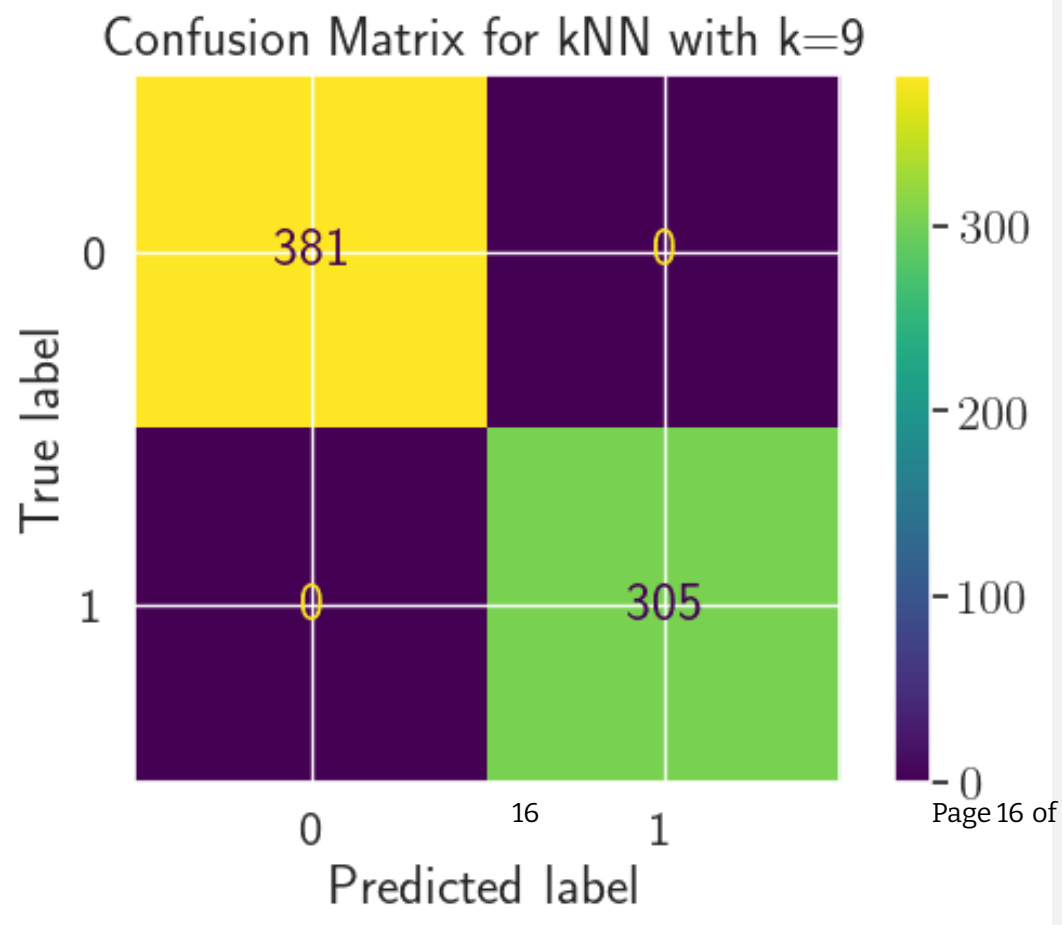
3.3 Use the optimal value k^* to compute performance measures and summarize them in the table.

Answer:

Cell 18

```
print("Selecting optimal k value as k=9.")

# Selecting k=9 to compute performance measures
classifier: KNeighborsClassifier = KNeighborsClassifier(
    n_neighbors=9, p=2, metric="euclidean"
)
classifier.fit(x_train_sc, y_train)
y_pred: np.ndarray = classifier.predict(x_test_sc)

# Create a confusion matrix to get metrics.
cm: np.array = confusion_matrix(y_test, y_pred)

# Create and show the confusion matrix plot.
disp: ConfusionMatrixDisplay = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.gcf().subplots_adjust(bottom=0.2)
plt.title("Confusion Matrix for kNN with k=9")
plt.show()

tn, fp, fn, tp = cm.ravel()

# Create a truth statistics table from kNN metrics.
knn_truth_statistics: DataFrame = get_truth_statistics_df(
    accuracy=accuracy_score(y_test, y_pred),
    true_positives=tp,
    false_positives=fp,
    true_negatives=tn,
    false_negatives=fn,
)
```

```
Selecting optimal k value as k=9.
```



Cell 19

```
# Build latex knn truth statistics  table to render in the document.
knn_truth_statistics_table = create_latex_table(
    knn_truth_statistics,
    label="tab:question3_3",
    caption="Question 3.3"
)
Latex(knn_truth_statistics_table)
```

Table 4: Question 3.3

	TP	FP	TN	FN	accuracy	TPR	TNR
0	305	0	381	0	1.0000	1.0000	1.0000

3.4 Is your k-NN classifier better than your simple classifier for any of the measures from the previous table?

Answer:

Yes the k-NN classifier performed better for all measures with 100% accuracy.

3.5 Consider a bill x that contains the last 4 digits of your BUID as feature values. What is the class label predicted for this bill by your simple classifier? What is the label for this bill predicted by k-NN using the best k^*?

Answer:

Cell 20

```
# Scale this new dataset.
bu_test_sc: np.ndarray = scaler.transform(bu_id_df)
bu_pred_knn: np.ndarray = classifier.predict(bu_test_sc).astype(int)

bu_bill_knn: str = "legitimate" if bu_pred_knn == 0 else "counterfeit"
print(f"kNN predicted my BUID to be a {bu_bill_knn} banknote!")

bu_pred_simple = bu_id_df.apply(simple_classifier, axis=1).iloc[0]
bu_bill_simple: str = "legitimate" if bu_pred_simple == 0 else "counterfeit"
print(
    f"Simple classifier predicted my BUID"
    f" to be a {bu_bill_simple} banknote!"
)
```

```
kNN predicted my BUID to be a legitimate banknote!
Simple classifier predicted my BUID to be a legitimate banknote!
```

4 Question 4: One of the fundamental questions in machine learning is "feature selection". We try to come up with a least number of features and still retain good accuracy. The natural question is whether some of the features are important or can be dropped.

4.1 Take your best value k^*. For each of the four features $f_1, \ldots, f_4$, drop that feature from both X_{train} and X_{test}. Train your classifier on the "truncated" X_{train} and predict labels on X_{test} using just 3 remaining features. You will repeat this for 4 cases: (1) just f_1 missing, (2) just f_2 (3) just f_3 missing and (4) just f_4 is missing. Compute the accuracy for each of these scenarios.

Answer:

Cell 21

```
# Compute the accuracy with optimal k when each feature is missing.
k: int = 9
f1_missing_accuracy: float = knn_accuracy(
    x_train=x_train.drop([COL_F1], axis=1),
    x_test=x_test.drop([COL_F1], axis=1),
    y_train=y_train,
    y_test=y_test,
    k=k
)
print(
    f"kNN accuracy when f1 is missing "
    f"{f1_missing_accuracy}."
)

f2_missing_accuracy: float = knn_accuracy(
    x_train=x_train.drop([COL_F2], axis=1),
    x_test=x_test.drop([COL_F2], axis=1),
    y_train=y_train,
    y_test=y_test,
    k=k
)
print(
    f"kNN accuracy when f2 is missing "
    f"{f2_missing_accuracy}."
)

f3_missing_accuracy: float = knn_accuracy(
    x_train=x_train.drop([COL_F3], axis=1),
    x_test=x_test.drop([COL_F3], axis=1),
    y_train=y_train,
    y_test=y_test,
    k=k
)
print(
    f"kNN accuracy when f3 is missing "
    f"{f3_missing_accuracy}."
)

f4_missing_accuracy: float = knn_accuracy(
    x_train=x_train.drop([COL_F4], axis=1),
    x_test=x_test.drop([COL_F4], axis=1),
    y_train=y_train,
    y_test=y_test,
    k=k
)
print(
    f"kNN accuracy when f4 is missing "
    f"{f4_missing_accuracy}."
)
```

```
kNN accuracy when f1 is missing 0.9416909620991254.
kNN accuracy when f2 is missing 0.9664723032069971.
kNN accuracy when f3 is missing 0.9591836734693877.
kNN accuracy when f4 is missing 0.9985422740524781.
```

4.2 Did accuracy increase in any of the 4 cases compared with accuracy when all 4 features are used?

Answer:

Accuracy decreased slightly for each case. k-NN accuracy was 100%, so they could only have been equal to or worse.

4.3 Which feature, when removed, contributed the most to loss of accuracy?

Answer:

When f_1 was removed, the accuracy dropped the most by about 6%

4.4 Which feature, when removed, contributed the least to loss of accuracy?

Answer:

When f_4 was removed, the accuracy dropped the least, still remaining above 99%. This aligns with the observation from earlier about entropy not having much of an effect on the pairwise differences.

5 Question 5: (use logistic regression classifier using sklearn library)

5.1 Use the same X_{train} and X_{test} as before. Train your logistic regression classifier on X_{train} and compute its accuracy for X_{test}.

Answer:

Cell 22

```
# Train on the dataset using logistic regression classifier.
logistic_regression: LogisticRegression = LogisticRegression()
logistic_regression.fit(x_train, y_train)
y_pred: np.ndarray = logistic_regression.predict(x_test)

log_reg_acc: float = accuracy_score(y_test, y_pred)

print(
    f"Accuracy for the logistic "
    f"regression classifier: {log_reg_acc}"
)
```

```
Accuracy for the logistic regression classifier: 0.9912536443148688
```

5.2 Summarize your performance measures in the table.

Answer:

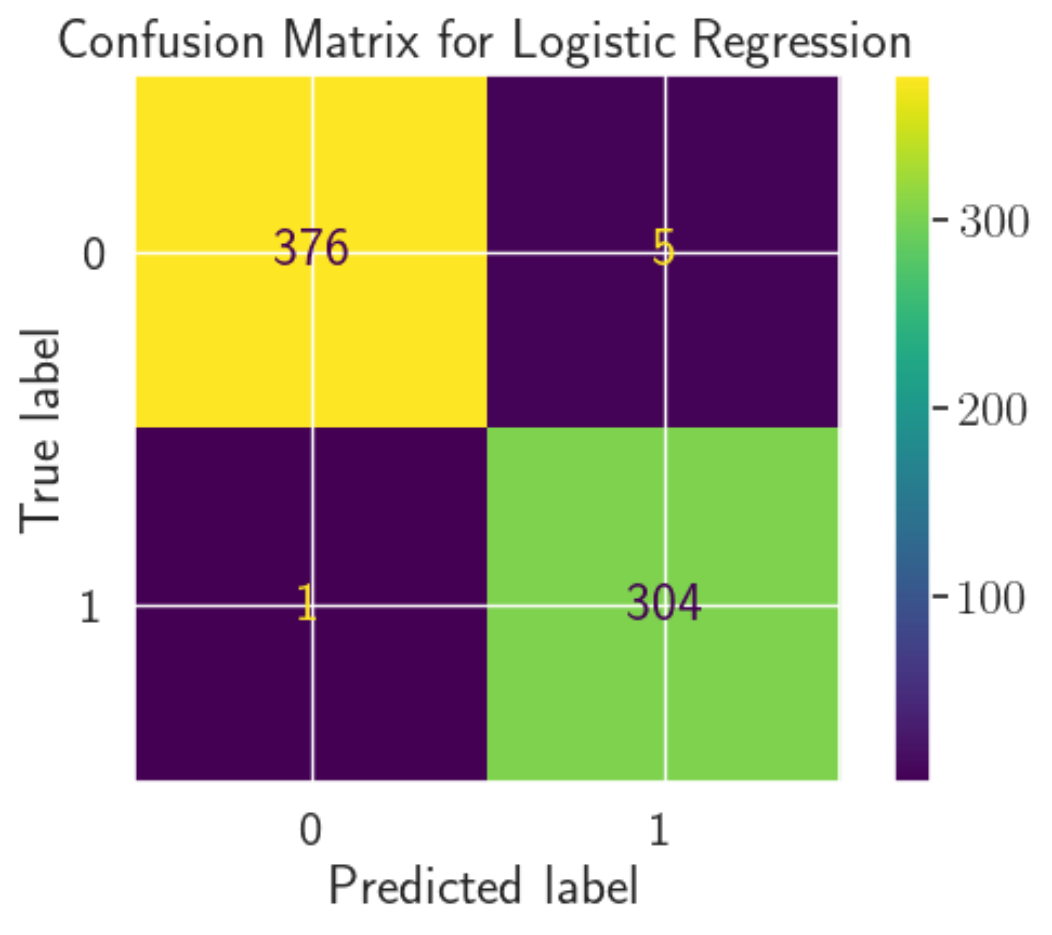
Cell 22

```
# Train on the dataset using logistic regression classifier.
logistic_regression: LogisticRegression = LogisticRegression()
logistic_regression.fit(x_train, y_train)
y_pred: np.ndarray = logistic_regression.predict(x_test)

log_reg_acc: float = accuracy_score(y_test, y_pred)

print(
    f"Accuracy for the logistic "
    f"regression classifier: {log_reg_acc}"
)
```

```
Accuracy for the logistic regression classifier: 0.9912536443148688
```

Cell 23

```
# Create a confusion matrix to get metrics.
cm: np.array = confusion_matrix(y_test, y_pred)

# Create and show the confusion matrix plot.
disp: ConfusionMatrixDisplay = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.gcf().subplots_adjust(bottom=0.2)
plt.title("Confusion Matrix for Logistic Regression")
save_figures("logistic_regression", artifacts)
plt.show()
```



Cell 24

```
# Get truth stats from confusion matrix.
tn, fp, fn, tp = cm.ravel()

# Create a truth statistics table from logistic regression metrics.
log_reg_truth_statistics: DataFrame = get_truth_statistics_df(
    accuracy=log_reg_acc,
    true_positives=tp,
    false_positives=fp,
    true_negatives=tn,
    false_negatives=fn,
)

# Build latex logistic regression truth statistics table to render in the document.
log_reg_truth_statistics_table = create_latex_table(
    log_reg_truth_statistics,
    label="tab:question5_2",
    caption="Question 5.2"
)
Latex(log_reg_truth_statistics_table)
```

Table 5: Question 5.2

	TP	FP	TN	FN	accuracy	TPR	TNR
0	304	5	376	1	0.9913	0.9967	0.9869

5.3 Is your logistic regression better than your simple classifier for any of the measures from the previous table?

Answer:

5.4 Is your logistic regression better than you k-NN classifier (using the best k^*) for any of the measures from the previous table?

Answer:

Yes, the logistic regression performed better at avoiding false positives much better than the simple classifier. The true positives were slightly lower than the simple classifier.

5.5 Consider a bill x that contains the last 4 digits of your BUID as feature values. What is the class label predicted for this bill x by logistic regression? Is it the same label as predicted by k-NN?

Answer:

Cell 25

```
# Predict BUID using logistic regression.
bu_pred_log_reg: np.ndarray = logistic_regression.predict(bu_id_df).astype(int)

bu_bill_log_reg: str = \
    "legitimate" if bu_pred_log_reg == 0 else "counterfeit"
print(
    f"Logistic regression predicted my "
    f"BUID to be a {bu_bill_log_reg} banknote!"
)
```

```
Logistic regression predicted my BUID to be a legitimate banknote!
```

6 Question 6: We will investigate change in accuracy when removing one feature.

This is similar to question 4, but now we use logistic regression.

Answer:

6.1 For each of the four features $f_1, \ldots, f_4$ drop that feature from both X_{train} and X_{test}. Train your logistic regression classifier on the "truncated" X_{train} and predict labels on X_{test} using just 3 remaining features. You will repeat this for 4 cases: (1) just f_1 is missing, (2) just f_2 is missing, (3) just f_3 is missing, and (4) just f_4 is missing. Compute the accuracy for each of these scenarios.

Answer:

Cell 26

```
# Compute the accuracy with when each feature is missing.
f1_missing_accuracy: float = log_reg_accuracy(
    x_train=x_train.drop([COL_F1], axis=1),
    x_test=x_test.drop([COL_F1], axis=1),
    y_train=y_train,
    y_test=y_test,
)
print(
    f"logistic regression accuracy when f1 is missing "
    f"{f1_missing_accuracy}."
)

f2_missing_accuracy: float = log_reg_accuracy(
    x_train=x_train.drop([COL_F2], axis=1),
    x_test=x_test.drop([COL_F2], axis=1),
    y_train=y_train,
    y_test=y_test,
)
print(
    f"logistic regression when f2 is missing "
    f"{f2_missing_accuracy}."
)

f3_missing_accuracy: float = log_reg_accuracy(
    x_train=x_train.drop([COL_F3], axis=1),
    x_test=x_test.drop([COL_F3], axis=1),
    y_train=y_train,
    y_test=y_test,
)
print(
    f"logistic regression accuracy when f3 is missing "
    f"{f3_missing_accuracy}."
)

f4_missing_accuracy: float = log_reg_accuracy(
    x_train=x_train.drop([COL_F4], axis=1),
    x_test=x_test.drop([COL_F4], axis=1),
    y_train=y_train,
    y_test=y_test,
)
print(
    f"logistic regression accuracy when f4 is missing "
    f"{f4_missing_accuracy}."
)
```

```
logistic regression accuracy when f1 is missing 0.7988338192419825.
logistic regression when f2 is missing 0.9052478134110787.
logistic regression accuracy when f3 is missing 0.8760932944606414.
logistic regression accuracy when f4 is missing 0.9912536443148688.
```

6.2 Did accuracy increase in any of the 4 cases compared with accuracy when all 4 features are used?

Answer:

The accuracy didn't increase when dropping any feature, but when dropping f_4, there was no change in accuracy.

6.3 Which feature, when removed, contributed the most to loss of accuracy?

Answer:

Dropping f_1 had the most loss of accuracy going down to 79%.

6.4 Which feature, when removed, contributed the least to loss of accuracy?

Answer:

When dropping f_4, there was no change in accuracy.

6.5 Is relative significance of features the same as you obtained using k-NN?

Answer:

Yes, all fixtures $f_1, \ldots, f_4$ had the same impact relatively to k-NN. The impact of dropping the features had much more of an impact on logistic regression though.