Exercise set 2

Declaration of generative AI use

please note that I have used generative AI to do formatting for markdown cells (mostly for latex in q6) and commenting on my code. I have not used generative ai for the creation of any of my code or text.

In [ ]:
```
### Import statements
import numpy as np
import pandas as pd
import statsmodels.api as sm
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import GaussianNB
from sklearn.dummy import DummyClassifier
from sklearn.preprocessing import PolynomialFeatures
import warnings
warnings.filterwarnings('ignore')
```

Problem 8

In [ ]:
```
### problem 8, task ai.

spam_test = pd.read_csv('data_E2/spam_test.csv')
spam_train = pd.read_csv('data_E2/spam_train.csv')
response_train = spam_train['SPAM']
response_test = spam_test['SPAM']
spam_train = spam_train.drop('SPAM', axis=1)
spam_test = spam_test.drop('SPAM', axis=1)
spam_train = sm.add_constant(spam_train)
spam_test = sm.add_constant(spam_test)

model = sm.Logit(response_train, spam_train)

result = model.fit()
print("coefficients:")
print(result.params[1:])
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.275032
         Iterations: 35
coefficients:
MISSING_FROM              -7.062601
FROM_ADDR_WS             -27.003504
TVD_SPACE_RATIO           19.877574
LOTS_OF_MONEY             38.589734
T_FILL_THIS_FORM_SHORT    41.764697
dtype: float64
```

In [ ]:
```
### problem 8, task aii.

# Predict the probabilities for the training set.
```

```
response_train_pred = result.predict(spam_train)
# Binarize the predicted probabilities: 1 if >= 0.5, otherwise 0.
response_train_pred_vals = np.array(response_train_pred)
response_train_pred_vals[response_train_pred_vals >= 0.5] = 1
response_train_pred_vals[response_train_pred_vals < 0.5] = 0

# Small constant to prevent log(0).
epsilon = 1e-15

# Calculate perplexity from the log likelihood.
loglikelihood = ((response_train_pred) * (response_train) +
                 (1 - response_train_pred) * (1 - response_train) + epsilon)
perplexity = np.exp(-np.mean(np.log(loglikelihood)))

print("training data accuracy:")
print(accuracy_score(response_train, response_train_pred_vals))
print("training data perplexity:")
print(perplexity)

# Repeat the process for the test data.
response_pred = result.predict(spam_test)
response_pred_vals = np.array(response_pred)
response_pred_vals[response_pred >= 0.5] = 1
response_pred_vals[response_pred < 0.5] = 0

loglikelihood = ((response_pred) * (response_test) +
                 (1 - response_pred) * (1 - response_test) + epsilon)
perplexity = np.exp(-np.mean(np.log(loglikelihood)))

print("testing data accuracy:")
print(accuracy_score(response_test, response_pred_vals))
print("testing data perplexity:")
print(perplexity)
```

```
training data accuracy:
0.88
training data perplexity:
1.3165728804311108
testing data accuracy:
0.88
testing data perplexity:
2.1872171476770728
```

problem 8, task aiii.

$$P(Y = 1 \mid x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n)}}$$

Here:

- $P(Y = 1 \mid x)$ is the probability that the outcome Y is 1 given the covariate vector x .
- e is the base of the natural logarithm.
- $\beta_0, \beta_1, \beta_2, \ldots, \beta_n$ are the coefficients of the logistic regression model.
- $x_1, x_2, \ldots, x_n$ are the elements of the covariate vector x.
- The expression $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$ represents the linear combination of the covariates weighted by their respective coefficients.

In [ ]:
```
### problem 8, task b.

# Define a range of values for the regularization
# parameter C in logistic regression.
c_vals = np.logspace(-3, 3, 7)
best_v_val = None
best_accuracy = 0

# Iterate over different values of C to find the
# one that gives the best accuracy.
for c in c_vals:
    # Initialise and train the model on the training dataset.
    model = LogisticRegression(penalty='l1', C=c,
                               solver='liblinear')
    model.fit(spam_train, response_train)
    # Predict on the test set and calculate accuracy.
    pred = model.predict(spam_test)
    accuracy = accuracy_score(response_test, pred)
    # Update the best C value necessary
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_c_val = c

# Train a new model using the best C value found.
best_model = LogisticRegression(penalty='l1', C=best_c_val,
                                solver='liblinear')
best_model.fit(spam_train, response_train)

# Evaluate the best model's performance on the test set.
y_pred = best_model.predict(spam_test)
accuracy = "{:.4f}".format(accuracy_score(response_test, y_pred))
# Calculate perplexity for the best model.
perplexity = np.exp(-np.mean(np.log(
    best_model.predict_proba(spam_test)
    [np.arange(len(response_test)), response_test])))
# Baseline accuracy for comparison (from last question)
accuracy_base = "{:.4f}".format(accuracy_score(response_test,
                                               response_pred_vals))

print("Best C:", best_c_val)
print("Coefficients:", best_model.coef_[0])
print("Baseline Accuracy:", accuracy_base)
print("Accuracy with lasso:", accuracy)
print("Perplexity:", perplexity)
```

```
Best C: 10.0
Coefficients: [-3.68993091  0.         -2.07387487  0.          3.62870203  6.778
57594]
Baseline Accuracy: 0.8800
Accuracy with lasso: 0.8800
Perplexity: 1.40673495810779
```

problem 8, task b (written).

the predicted class probabilities from the unregularized regressor in Task a are much higher than they are from the regulated task probablilities in task b

Problem 9

Assume $f_k(x)$ is gaussian, we have

[1] $f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{1}{2\sigma_k^2}(x-\mu_k)^2\right)$ {equation 4.16 from book}

where

μ_k is the mean of the k^{th} class

σ_k is the variance of the k^{th} class

We also have the posterior probability $P_k(x)$ which gives the probability an observation $X=x$ is in the k^{th} class

[2] $P_k(x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}$

Where

π_k is the overall prior probability that a randomly chosen observation is in the k^{th} class

$f_k(x)$ is [1]

Now, solving for δ_k

$$\delta_k(x) = \log(P_k(x))$$

$$= \log\left(\frac{\pi_k f_k(x)}{\sum_{l=1}^{k} \pi_l f_l(x)}\right) = \log(\pi_k f_k(x)) - \log\left(\sum_{l=1}^{k} \pi_l f_l(x)\right) \text{ {by 2}}$$

$$= \log(\pi_k) + \log(f_k(x)) - \log\left(\pi_1 f_1(x) + \pi_2 f_2(x)\right)$$

$$= \log(\pi_k) + \log\left(\frac{1}{\sqrt{2\pi}\sigma_k}\right) + \log\left(e^{\left(\frac{1}{2\sigma_k^2}(x-\mu_k)^2\right)}\right) - \log\left(\pi_1 f_1(x) + \pi_2 f_2(x)\right)$$

$$= \log(\pi_k) + \log\left(\frac{1}{\sqrt{2\pi}\sigma_k}\right) - \log\left(\pi_1 f_1(x) + \pi_2 f_2(x)\right) + \frac{1}{2\sigma_k^2}(x-\mu_k)^2$$

$$= \log(\pi_k) - \log\left(\sqrt{2\pi}\sigma_k\right) - \log\left(\pi_1 f_1(x) + \pi_2 f_2(x)\right) + \frac{1}{2\sigma_k^2}(x-\mu_k)^2$$

$$\delta_k(x) = \log(\pi_k) - \log\left(\sqrt{2\pi}\right) - \log(\sigma_k) - \log\left(\pi_1 f_1(x) + \pi_2 f_2(x)\right) + \frac{1}{2\sigma_k^2}(x-\mu_k)^2$$

Now, to see when QDA is linear, we need to find the bays decision boundry & show for which values σ_k make the coefficient of x^2 term 0

We have bayes decision boundry at point

$$
\begin{aligned}
0 &= \delta_1(x) - \delta_2(x) \\
&= \log(\pi_1) - \log(\sqrt{2\pi}) - \log(\sigma_1) - \log\left(\pi_1 f_1(x) + \pi_2 f_2(x)\right) + \frac{1}{2\sigma_1^2}(x-\mu_1)^2 \\
&\quad - \left(\log(\pi_2) - \log(\sqrt{2\pi}) - \log(\sigma_2) - \log\left(\pi_1 f_1(x) + \pi_2 f_2(x)\right) + \frac{1}{2\sigma_2^2}(x-\mu_2)^2\right) \\
&= \log(\pi_1) - \log(\sigma_1) + \frac{1}{2\sigma_1^2}(x-\mu_1)^2 \\
&\quad - \left(\log(\pi_2) - \log(\sigma_2) + \frac{1}{2\sigma_2^2}(x-\mu_2)^2\right) \\
&= \log\left(\frac{\pi_1\sigma_2}{\pi_2\sigma_1}\right) + \frac{2\sigma_2^2(x-\mu_1)^2 - 2\sigma_2^2(x-\mu_2)^2}{4\sigma_1^2\cdot\sigma_2^2} \\
&= \log\left(\frac{\pi_1\sigma_2}{\pi_2\sigma_1}\right) + \frac{\sigma_2^2(x^2-2x\mu_1+\mu_1^2) - \sigma_1^2(x^2-2x\mu_2+\mu_2^2)}{2\sigma_1^2\cdot\sigma_2^2} \\
&= \log\left(\frac{\pi_1\sigma_2}{\pi_2\sigma_1}\right) + \frac{x^2(\sigma_2^2-\sigma_1^2)}{2\sigma_1^2\cdot\sigma_2^2} - \frac{x(\sigma_2^2\mu_1+\sigma_1^2\mu_2)}{\sigma_1^2\cdot\sigma_2^2} + \frac{\sigma_2^2\mu_1^2+\sigma_1^2\mu_2^2}{\sigma_1^2\cdot\sigma_2^2}
\end{aligned}
$$

Solving just to find when $x^2 = 0$

$$
\begin{aligned}
0 &= \frac{x^2(\sigma_2^2-\sigma_1^2)}{2\sigma_1^2\cdot\sigma_2^2} \\
0 &= x^2(\sigma_2^2-\sigma_1^2) \\
\sigma_2^2 &= \sigma_1^2
\end{aligned}
$$

Thus QDA is only linear when $\sigma_1^2 = \sigma_2^2$

$\therefore$ QDA is non linear if $\sigma_1^2 \neq \sigma_2^2$

Problem 10

In [ ]:
```
# Load training and testing data from CSV files.
train = pd.read_csv('data_E2/penguins_train.csv')
test = pd.read_csv('data_E2/penguins_test.csv')

# Recode species column to binary: 'Adelie' or 'notAdelie'.
train_res = train['species'].apply(lambda x: 'Adelie'
                                   if x == 'Adelie' else 'notAdelie')
test_res  = test['species'].apply(lambda x: 'Adelie'
                                   if x == 'Adelie' else 'notAdelie')

# Separate the data into Adelie and not Adelie for the training set.
adelie_data = train[train['species'] == 'Adelie']
not_adelie_data = train[train['species'] != 'Adelie']
```

```
# Remove the species column as it's now encoded in train_res and test_res.
train = train.drop('species', axis=1)
test = test.drop('species', axis=1)

# Calculate descriptive statistics for each feature by species.
stats = []
for feature in ['bill_length_mm', 'bill_depth_mm',
                'flipper_length_mm', 'body_mass_g']:
    stats.append({
        'Feature': feature,
        'Mean_Ad': adelie_data[feature].mean(),
        'StdDev_Ad': adelie_data[feature].std(),
        'Mean_notAd': not_adelie_data[feature].mean(),
        'StdDev_notAd': not_adelie_data[feature].std()
    })

# Convert stats to a DataFrame and set 'Feature' as the index.
stats = pd.DataFrame(stats).set_index('Feature')
print(stats)

# Calculate the probability of being Adelie or not with Laplace smoothing.
num_ad = len(adelie_data)
num_not_ad = len(not_adelie_data)
prob_ad = (num_ad + 1) / (num_ad + num_not_ad + 2)
prob_not_ad = (num_not_ad + 1) / (num_ad + num_not_ad + 2)

# Print the calculated probabilities.
print("\nAdelie probability:", prob_ad)
print("not Adelie probability:", prob_not_ad)
```

```
                   Mean_Ad   StdDev_Ad  Mean_notAd  StdDev_notAd
Feature
bill_length_mm      38.124    2.781528      47.818      3.599472
bill_depth_mm       18.336    1.204118      15.890      1.965441
flipper_length_mm  188.880    6.320074     211.300     11.792855
body_mass_g       3576.000  461.343148    4657.000    787.531017

Adelie probability: 0.33766233766233766
not Adelie probability: 0.6623376623376623
```

problem 10, task b

$$\hat{p} = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(x_i - \mu_i)^2}{2\sigma_i^2}\right)$$

where:

- σ_i^2 is the standard deviation of the ith attribute in the adelle class
- μ_i is the mean of the ith attribute in the adelle class

In [ ]:

```
### Problem 10, task c:

def naive_bays_classifier(x, stats):
    # Initialize probabilities with the prior probabilities of each class.
    p_ad = prob_ad
    p_not = prob_not_ad
    # Calculate the likelihood of the data point for each feature given the
```

```
    # class using Gaussian Naive Bayes formula for classes.
    for feature in ['bill_length_mm', 'bill_depth_mm',
                    'flipper_length_mm', 'body_mass_g']:
        p_ad *= (1/(np.sqrt(2 * np.pi) * stats.loc[feature, 'StdDev_Ad']))*\
                 np.exp(-((x[feature] - stats.loc[feature, 'Mean_Ad'])**2) /
                          (2 * stats.loc[feature, 'StdDev_Ad']**2))
        p_not *= (1/(np.sqrt(2 * np.pi) * stats.loc[feature, 'StdDev_notAd']))*\
                  np.exp(-((x[feature] - stats.loc[feature, 'Mean_notAd'])**2) /
                           (2 * stats.loc[feature, 'StdDev_notAd']**2))
    # return the class with the higher probability.
    return 'Adelie' if p_ad > p_not else 'notAdelie'

# Apply the classifier and the first three predictions.
pred = test.apply(lambda x: naive_bays_classifier(x, stats), axis=1)
print(pred[0:3])
```

```
0    Adelie
1    Adelie
2    Adelie
dtype: object
```

Problem 11

Problem 11 task a:

- According to the authors, is discriminative learning better than generative learning?
  - No, the autors argue that in contrast to the widely-held belief that "in almost all situations, discriminative learning is better than generative learning", as the training set size increases, one algorithm will imarge to be a better fit than the other.

Problem 11 task b:

- Ng and Jordan denote by $hGen$ and $hDis$ two models chosen by optimizing different objectives. Which two families do the authors discuss, and what are the ($hGen$, $hDis$) pairs for those models? What objectives are being optimised?
  - the two families tha that the authors discuss are Gaussian/Normal and multinomial distributions.
  - the ($hGen$, $hDis$) pairs For Gaussian/Normal distributions are Normal Discriminant Analysis ($hGen$) and logistic regression ($hDis$).
  - the ($hGen$, $hDis$) pairs For multinomial distributions are Naive Bayes classifier ($hGen$) and logistic regression ($hDis$).
  - hGen optimises the joint liklyhood of inputs and labels
  - hDis either optimises the conditional likelihood p(y|x) or the 0-1 training error

Problem 11 task c:

- Study Figure 1 in the paper. Explain what it suggests (see the last paragraph of the Introduction). Reflect on what this means for the families in Task b.
  - figure 1 suggests that while initially, the generative model (naive Bayes) performs better due to its faster approach to its asymptotic error, as the

number of training examples increases, the discriminative model (logistic regression) catches up and surpasses the generative model due to its lower asymptotic error.

- for the families in task b, this means that the size of hte dataset is an important metric for deciding which model to use.

Problem 12

problem 12 task a.

No, the naieves bayes assumption does not hold as Naive Bayes assumes that x1 and x2 are independent. hoewever the probability of y is defined as a function that includes an interaction term between x1 and x2, thus x1 and x2 are not independant.

In [ ]:
```
### problem 12 task b.

# Define a range of dataset sizes as powers of 2 from 2^3 to 2^12.             f
n_values = [2**i for i in range(3, 13)]

# Initialize dataframes to hold accuracy and perplexity results
# for different classifiers.
out_table = pd.DataFrame(columns=['n', 'NB', 'LR', 'LRi',
                                  'OptimalBays', 'Dummy'])
out_table['n'] = n_values
out_table_accuracy = out_table.set_index('n')
out_table_perplexity = out_table.set_index('n')

# Load the test set and separate features from the target variable.
test = pd.read_csv('data_E2/toy_test.csv')
test_res = test['y']
test = test.drop('y', axis=1)

# Iterate over each dataset size to train models and evaluate performance.
for n in n_values:
    # Load the training set corresponding to the current size.
    train = pd.read_csv('data_E2/toy_train_' + str(n) + '.csv')
    train_res = train['y']
    train = train.drop('y', axis=1)

    # Train and evaluate a Gaussian Naive Bayes classifier.
    model = GaussianNB()
    model.fit(train, train_res)
    pred = model.predict(test)
    loglikelihood = model.predict_proba(test)[np.arange(len(test_res)), test_res
    perplexity = np.exp(-np.mean(np.log(loglikelihood)))

    # Store the accuracy and perplexity in the respective dataframes.
    out_table_accuracy.loc[n, 'NB'] = accuracy_score(test_res, pred)
    out_table_perplexity.loc[n, 'NB'] = perplexity

    # Train and evaluate a Logistic Regression classifier without regularization
    model = LogisticRegression(penalty='none')
    model.fit(train, train_res)
    pred = model.predict(test)
    loglikelihood = model.predict_proba(test)[np.arange(len(test_res)), test_res
```

```
    perplexity = np.exp(-np.mean(np.log(loglikelihood)))
    out_table_accuracy.loc[n, 'LR'] = accuracy_score(test_res, pred)
    out_table_perplexity.loc[n, 'LR'] = perplexity

    # Train and evaluate a Logistic Regression classifier with L1 regularization
    model = LogisticRegression(penalty='l1', C=1, solver='liblinear')
    model.fit(train, train_res)
    pred = model.predict(test)
    loglikelihood = model.predict_proba(test)[np.arange(len(test_res)), test_res
    perplexity = np.exp(-np.mean(np.log(loglikelihood)))
    out_table_accuracy.loc[n, 'LRi'] = accuracy_score(test_res, pred)
    out_table_perplexity.loc[n, 'LRi'] = perplexity

    # Define the Optimal Bayes classifier function using the given equation.
    def optimal_bayes_classifier(x1, x2):
        t = -0.5 - x1 + 1.5 * x2 + (x1 * x2) / 3
        return 1 / (1 + np.exp(-t))

    # Apply the Optimal Bayes classifier to the test set.
    probs = test.apply(lambda x: optimal_bayes_classifier(x['x1'], x['x2']),
                       axis=1)
    pred = (probs >= 0.5).astype(int)

    # Calculate and store perplexity and accuracy for the Optimal Bayes classifi
    loglikelihood = ((probs) * (test_res) +
                     (1 - probs) * (1 - test_res) + epsilon)
    perplexity = np.exp(-np.mean(np.log(loglikelihood)))
    out_table_perplexity.loc[n, 'OptimalBays'] = perplexity
    out_table_accuracy.loc[n, 'OptimalBays'] = accuracy_score(test_res, pred)

    # Train and evaluate a Dummy classifier that predicts the majority class.
    model = DummyClassifier(strategy='prior')
    model.fit(train, train_res)
    pred = model.predict(test)
    probs = model.predict_proba(test)[:,1]

    # Calculate and store perplexity and accuracy for the Dummy classifier.
    loglikelihood = ((probs) * (test_res) +
                     (1 - probs) * (1 - test_res) + epsilon)
    perplexity = np.exp(-np.mean(np.log(loglikelihood)))
    out_table_perplexity.loc[n, 'Dummy'] = perplexity
    out_table_accuracy.loc[n, 'Dummy'] = accuracy_score(test_res, pred)

# Output the accuracy and perplexity tables for comparison of classifiers.
print("accuracy table:")
print(out_table_accuracy)
print("\nperplexity table:")
print(out_table_perplexity)
```

```
accuracy table:
          NB      LR     LRi OptimalBays   Dummy
n
8     0.5973  0.6837  0.6957      0.7572  0.5688
16    0.6537  0.6334  0.6277      0.7572  0.4312
32    0.7037  0.7337  0.7365      0.7572  0.5688
64    0.6843  0.7531  0.7506      0.7572  0.5688
128   0.7444  0.7533  0.7528      0.7572  0.5688
256   0.7507    0.75  0.7511      0.7572  0.5688
512   0.7525  0.7528  0.7532      0.7572  0.5688
1024  0.7476  0.7505  0.7506      0.7572  0.5688
2048  0.7518  0.7524  0.7527      0.7572  0.5688
4096  0.7513   0.751  0.7509      0.7572  0.5688

perplexity table:
             NB        LR       LRi OptimalBays     Dummy
n
8     81.029145  1.804528   1.88989     1.63518       2.0
16     2.557424  6.102342  2.221673     1.63518  2.139477
32     1.954326  1.717563  1.694754     1.63518       2.0
64     1.892499  1.653345  1.667628     1.63518  1.981256
128    1.665193  1.657166  1.652697     1.63518  1.983639
256    1.645215  1.645612   1.64588     1.63518  1.981105
512    1.656468  1.650604  1.649511     1.63518  1.981119
1024   1.653585  1.645227  1.645366     1.63518  1.984055
2048   1.654582   1.64527  1.645372     1.63518  1.981121
4096   1.648145  1.645596  1.645583     1.63518  1.981116
```

In [ ]:

```python
### Problem 12 task c.

# Load the training and testing datasets.
train = pd.read_csv('data_E2/toy_train_4096.csv')
test = pd.read_csv('data_E2/toy_test.csv')

# Prepare the features (x) and
# target (y) for the training set.
x_train = train.drop('y', axis=1)
res_train = train['y']

# Generate polynomial features with degree 2,
# excluding bias (intercept) term.
poly = PolynomialFeatures(degree=2, interaction_only=True,
                          include_bias=False)
x_train_poly = poly.fit_transform(x_train)

# Train a Logistic Regression model without
# regularization on polynomial features.
model = LogisticRegression(penalty='none')
model.fit(x_train_poly, res_train)

# Retrieve and print the feature names and
# corresponding coefficients from the model.
features = poly.get_feature_names_out(x_train.columns)
coefficients = pd.DataFrame({'Feature': features,
                             'Coefficient': model.coef_[0]})
coefficients = coefficients.set_index('Feature')
print(coefficients)
```

```
         Coefficient
Feature
x1         -1.029608
x2          1.448626
x1 x2       0.345481
```

Problem 12 Task C (Written Portion):

The regression coefficients:

- x1: -1.029608
- x2: 1.448626
- (x1 x2): 0.345481

are very close, but not identical, to the actual model coefficients:

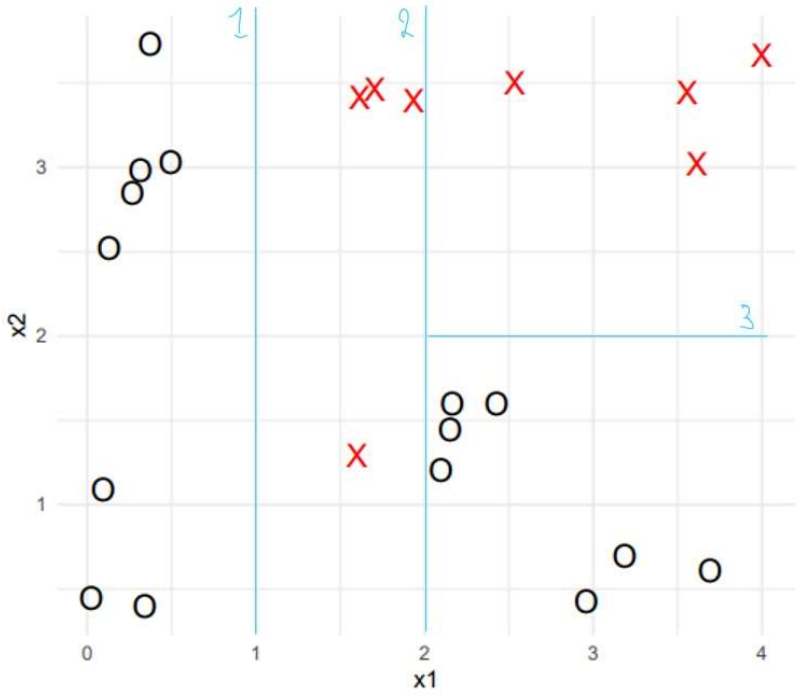
- x1: -1
- x2: 1.5
- (x1 x2): 0.333333

Model Analysis

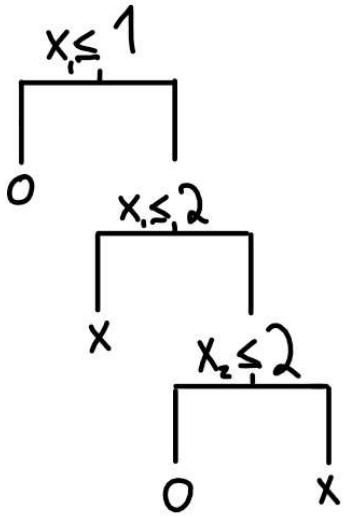
- **Which of the models above are probabilistic, discriminative, and generative?**
  - The actual model and the Optimal Naive Bayes are probabilistic.
  - The Naive Bayes model is generative.
  - The logistic regression models are discriminative.
  - The dummy model is a non-learning model, which could loosely be categorized as probabilistic.
- **How do accuracy and perplexity (log-likelihood) compare?**
  - We can see that as perplexity decreases, accuracy tends to increase.
- **Is there a relation to the insights from the previous problem?**
  - It is clear that with smaller datasets, the discriminative models tend to outperform the generative models. However, as the dataset size increases, the generative models begin to outperform the discriminative ones. This observation correlates with the discussion in Question 11.
- **Why does logistic regression with the interaction term perform so well for larger datasets?**
  - For larger datasets, logistic regression can estimate the actual model's coefficients with a high degree of accuracy by capturing the complex interactions more effectively.
- **Does your dummy classifier ever outperform other classifiers, or do different classifiers outperform the optimal Bayes classifier?**
  - No, the dummy classifier does not outperform other classifiers. Similarly, other classifiers do not outperform the optimal Bayes classifier.

Problem 13

sketch



Decision Tree:



We have

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

Where

G = Gini index
K = # of classes
$\hat{p}_{mk}$ = proportion of training observations in the m^{th} region from the k^{th} class

Split 1

$X_{x_1 \le 1} = 0 \quad O_{x_1 \le 1} = 8$

$X_{x_1 > 1} = 8 \quad O_{x_1 > 1} = 7$

Total = 23

$$G_0 = \frac{8}{23}\left(1-\frac{8}{23}\right) + \frac{15}{23}\left(1-\frac{15}{23}\right) = 2\left(\frac{8}{23}\cdot\frac{15}{23}\right)$$

$$G_1 = G_0 - \frac{8}{23}G_{x_1<1} - \frac{15}{23}G_{x_1\ge 1}$$

$$= 2\left(\frac{8}{23}\cdot\frac{15}{23}\right) - \frac{8}{23}\left(\frac{8}{8}\left(1-\frac{8}{8}\right) + \frac{0}{8}\left(1-\frac{8}{8}\right)\right) - \frac{15}{23}\left(\frac{7}{23}\left(1-\frac{7}{23}\right) + \frac{8}{23}\left(1-\frac{8}{23}\right)\right)$$

$G_1 = 0.13$

Split 2

$X_{x_1 < 2} = 4 \quad O_{x_1 < 2} = 0$

$X_{x_1 \ge 2} = 4 \quad O_{x_1 \ge 2} = 7$

Total = 15

$$G_0 = \frac{8}{15}\left(1-\frac{8}{15}\right) + \frac{7}{15}\left(1-\frac{7}{15}\right)$$

$$G_2 = G_0 - \frac{4}{15}G_{x_1<2} - \frac{11}{15}G_{x_1\ge 2}$$

$$= \frac{8}{15}\left(1-\frac{8}{15}\right)+\frac{7}{15}\left(1-\frac{7}{15}\right)-\frac{4}{15}(0)$$

$$\frac{11}{15}\left(\frac{4}{15}\left(1-\frac{4}{15}\right)+\frac{7}{15}\left(1-\frac{7}{15}\right)\right)$$

$$G_2 = 0.16$$

Split 3

$$X_{x_2 \le 2} = 0 \qquad O_{x_2 \le 2} = 7$$

$$X_{x_2 > 2} = 4 \qquad O_{x_2 > 2} = 0$$

total = 11

$$G_0 = \frac{4}{11}\left(1-\frac{4}{11}\right)+\frac{7}{11}\left(1-\frac{7}{11}\right)$$

$$G_3 = G_0 - \frac{7}{11} G_{x_2 \le 0} - \frac{4}{11} G_{x_2 > 0}$$

$$= \frac{4}{11}\left(1-\frac{4}{11}\right)+\frac{7}{11}\left(1-\frac{7}{11}\right)-\frac{7}{11}(0)$$

$$-\frac{4}{11}(0)$$

$$G_3 = 0.463$$

Problem 14

Problem 14 task a

- the classification boundries for 1-NN are as follows
  - 4 as the class changes from +1 to -1.
  - 5.5 the class changes from -1 to +1.
  - 10.5 the class changes from +1 to -1.
  - 15.5 the class changes from -1 to +1.
  - 17 the class changes from +1 to -1.
- for 1NN error rate:
  - there are no errors
- the classification boundries for 3-NN are as follows
  - 10.5 as the class changes from +1 to -1.

- for 3NN error rate,
  - Point 4 (5.0) will be misclassified as +1 because its neighbors (3.0, 6.0, 8.0) include three +1s.
  - Similarly, point 11 (16.0) will be misclassified as -1 because its neighbors (13.0, 15.0, 18.0) include three -1s.
  - giving an error rate of 2/14 = 1/7

Problem 14 task b

- the choise of k in kNN will have an effect on how much the model will generalise the data.
- For small k:
  - the model will classify all areas around training data points as whatever its closest training data point is, thi will result in all predictions beside outliers and anomalies getting classified as the same class as the outlier.
  - for example, with k = 1, the model will classify any input as its closest neighbour. In the model trained on D, this would look like input 4.2 getting classfied as -1.
- For Large k:
  - with a large k, the model will smooth out the prediction over a larger area, this will ignore noise and lead to a more generalizable model. However it will miss and fine details of the data set.
  - For example, with k equal to the size of the training dataset, the classifier will predict the majority class for all points, ignoring the input features entirely.

Problem 15

Problem 15 task a:

by geometric intuition, we can see that the points from both classes can be separated with a vertical line between the x-coordinates of the class 1 and class -1 points (somewhere between 2 < x1 < 4).

to get the separating hyperplane with the largest margin on the data set we will find the median of the x coordinates closest to this separating hyperplane D and (E,F) (as E,F are both class -1 with the same x1 coordinate)

$$x1 = (2 + 4)/2$$

$$x1 = 3$$

thus the hyper plane has equation $x1 = 3.0$ and support vectors D,E,F

Problem 16

Problem 16 task a

Over these two weeks, I learned about discriminative and generative classifiers, how they work, and the differences between them. I also revisited k-nearest neighbor, decision trees, and SVM algorithms.

Unfortunatly I didn't really have time to digest the full content covered. I have been at a conference for the first week and a half and have only really had time in the last three days prior to submission to do the homework. I was lucky that most of the topics were things I already had a basic understanding of, which allowed me to quickly progress through the problems. For the next week's homework, I will have more time for studying, and I hope to re-read these chapters and get a better understanding of the topics.

Problem 16 task b.

about 15 hours