Homework 3 – Report

Programming Exercises

1. Binary Classification on Text Data

Download

```
#DOWNLOAD THE DATA
filepath_train = (r"C:\Users\Sofia Beyerlein\Desktop\Cornell Graduate\Applied Machine Learning\hw2\nlp-getting-started\train.
filepath_test = (r'C:\Users\Sofia Beyerlein\Desktop\Cornell Graduate\Applied Machine Learning\hw2\nlp-getting-started\test.cs

train = pd.read_csv(filepath_train)
test = pd.read_csv(filepath_test)
```

```
#SPLITTING THE DATA
#70% -> 5329/7613 and 30% -> 2284
training_set = train.sample(frac=0.7)
dev_set = train.drop(training_set.index)
```

Split training data

```
#SPLITTING THE DATA
#70% -> 5329/7613 and 30% -> 2284
training_set = train.sample(frac=0.7)
dev_set = train.drop(training_set.index)
```

Preprocess Data

```
def preprocess_data(df):
    words_to_remove = {'the', 'and', 'or'}
    #lowercase
    df['text'] = df['text'].apply(lambda x: x.lower())
    #remove @ and urls
    df['text'] = df['text'].apply(lambda x: re.sub(r'@\S+', '', x))
    #remove # and hashtags
    df['text'] = df['text'].apply(lambda x: re.sub(r'#\S+', '', x))
    #strip punctuation
    df['text'] = df['text'].apply(lambda x: x.translate(str.maketrans('', '', string.punctuation)))
    #strip the and or
    df['text'] = df['text'].apply(lambda x: ' '.join(word for word in x.split() if word not in words_to_remove))
    #lemmatize
    lemmatizer = nltk.WordNetLemmatizer()
    stop_words = set(stopwords.words('english'))

    def lemmatize_text(text):
        tokens = nltk.word_tokenize(text)
        lemmatized_tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
        return ' '.join(lemmatized_tokens)

    df['text'] = df['text'].apply(lemmatize_text)

    return df


preprocess_data(training_set)
preprocess_data(dev_set)
```

Bag of Words of Model

```
M = 3
vectorizer = CountVectorizer(binary=True, min_df=M)

train_vectors = vectorizer.fit_transform(training_set['text'])
dev_vectors = vectorizer.transform(dev_set['text'])
```

Part a: Bernoulli Naïve Bayes

```
#functions for the bernoulli naive bayes classifier with laplace smoothing
def calculate_class_priors(y):
    num_docs = len(y)
    class_priors = np.bincount(y) / num_docs
    return class_priors

def calculate_feature_probs(X, y, alpha=1):
    num_docs, num_features = X.shape
    feature_probs = np.zeros((2, num_features))

    for k in range(2):
        class_docs = X[y == k]
        feature_probs[k] = (class_docs.sum(axis=0) + alpha) / (class_docs.shape[0] + 2 * alpha)

    return feature_probs

def predict_log_proba(X, class_priors, feature_probs):
    num_docs, num_features = X.shape
    log_probs = np.zeros((num_docs, 2))

    for k in range(2):
        log_prob_k = np.log(class_priors[k])
        log_prob_x_given_k = X @ np.log(feature_probs[k]) + (1 - X) @ np.log(1 - feature_probs[k])
        log_probs[:, k] = log_prob_k + log_prob_x_given_k

    return log_probs

def predict(X, class_priors, feature_probs):
    log_probs = predict_log_proba(X, class_priors, feature_probs)
    return np.argmax(log_probs, axis=1)
```

```
#class prior calculations
class_priors = calculate_class_priors(training_set['target'].values)

#feature probablilities with laplace smoothing calculations
feature_probs = calculate_feature_probs(train_vectors.toarray(), training_set['target'].values, alpha=1)

#dev set predictions
dev_predictions = predict(dev_vectors.toarray(), class_priors, feature_probs)

#F1 score dd=ev set
f1 = f1_score(dev_set['target'], dev_predictions)
print(f"F1 Score: {f1}")
```

```
F1 Score: 0.7390562819783968
```

The F1 score of the development set was 0.73906

Part b: Model Comparison

- The F1 score on the L1 regression was 0.70699 while the naïve bayes had an F1 score of 0.72968. Therefore, the naïve bayes model performed best in predicting whether a tweet is a real disaster or not. The pros of using generative vs discriminative models are that generative models can manage missing data better and can recognize speech patterns because of its ability to infer. However, the cons are that it's more computationally expensive and that it can't classify data very well.
- The assumptions of naïve bayes are different from logistic regression because it assumes that features are independent and logistic regression assumes that the features have a linear relationship.

2. Gaussian Discriminant Analysis

```
#downloading the iris data
from sklearn import datasets
iris = datasets.load_iris ( as_frame = True )
iris_df = iris.frame

file_path = r"C:\Users\Sofia Beyerlein\Desktop\Cornell Graduate\Applied Machine Learning\hw3\iris_dataset.csv"
iris_df.to_csv(file_path, index=False)

df = pd.read_csv(file_path)
df
```

Splitting the data

```
#splitting the dataset into testing and training sets
#training: a, b, c, e
#testing: d

training_set = df.sample(frac=0.8)
dev_set = df.drop(training_set.index)
```

Part a

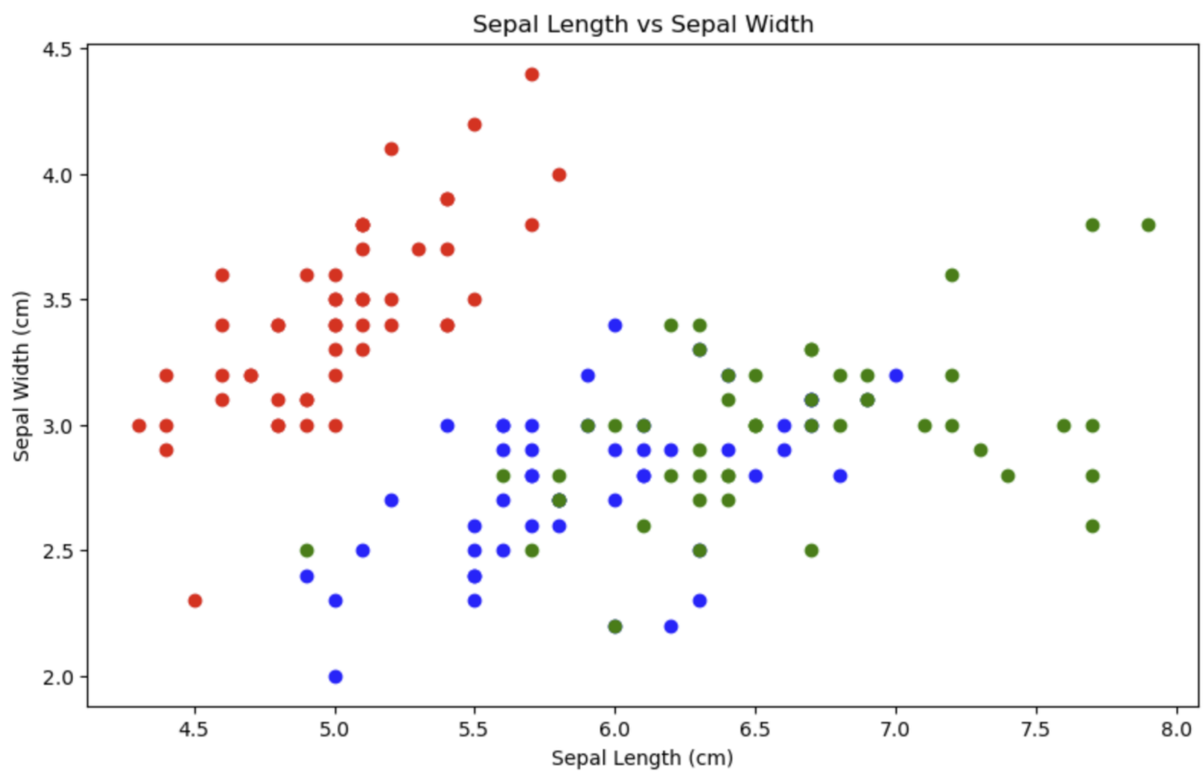
```
#using training data
#sepal length and sepal width
feature_1 = 'sepal length (cm)'
feature_2 = 'sepal width (cm)'

setosa = training_set[training_set['target'] == 0]
versicolor = training_set[training_set['target'] == 1]
virginica = training_set[training_set['target'] == 2]

plt.figure(figsize=(10, 6))
plt.scatter(setosa[feature_1], setosa[feature_2], color='red', label='Setosa')
plt.scatter(versicolor[feature_1], versicolor[feature_2], color='blue', label='Versicolor')
plt.scatter(virginica[feature_1], virginica[feature_2], color='green', label='Virginica')

plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.title('Sepal Length vs Sepal Width')

plt.show()
```



I notice that the Setosa class is pretty distinguished in terms of sepal length and sepal width ranging from width [2.8, 4.4] and sepal length [4.4, 5.6]. Meanwhile, Versicolor and Virginica tend to have similar ranges of sepal width around [2.3, 3.5] typically and while Versicolor has a smaller range of sepal length, there seems to be a substantial overlap between the sepal lengths of Versicolor and Virginica around [5.5, 7.0].

Part b

```
#use training set
#n_k number of observations in class k -> num in
#setosa, versiocolor, virginica
#N number of total observations in the data set

#these are the n_k
setosa_counts = len(setosa)
versicolor_counts = len(versicolor)
virginica_counts = len(virginica)

#N
N = len(training_set)

prior_probability = {
    "setosa" : (setosa_counts/N),
    "versicolor" : (versicolor_counts/N),
    "virginica" : (virginica_counts/N),
}

print(f"The prior probability of setosa is: {prior_probability['setosa']}")
print(f"The prior probability of setosa is: {prior_probability['versicolor']}")
print(f"The prior probability of setosa is: {prior_probability['virginica']}")
```

```
The prior probability of setosa is: 0.30833333333333335
The prior probability of setosa is: 0.35
The prior probability of setosa is: 0.3416666666666667
```

Part c

```
features = iris.feature_names
target = 'target'

class_stats = {}
for class_index in training_set[target].unique():
    class_data = training_set[training_set[target] == class_index][features]
    mean_vector = class_data.mean().values
    covariance_matrix = class_data.cov().values

    class_stats[class_index] = {
        'mean': mean_vector,
        'covariance': covariance_matrix
    }

#Print the matrix for each class
for class_index, stats in class_stats.items():
    class_name = iris.target_names[class_index]
    print(f"Class: {class_name}")
    print(f"Mean vector {class_name}:")
    print(stats['mean'])
    print(f"Covariance matrix {class_name}:")
    print(stats['covariance'])
    print()
```

```
Class: versicolor
Mean vector versicolor:
[5.96190476 2.78333333 4.27619048 1.32380952]
Covariance matrix versicolor:
[[0.25509872 0.07715447 0.17004646 0.04849013]
 [0.07715447 0.09800813 0.07642276 0.03894309]
 [0.17004646 0.07642276 0.2218583  0.06960511]
 [0.04849013 0.03894309 0.06960511 0.03746806]]

Class: setosa
Mean vector setosa:
[5.00540541 3.42162162 1.46756757 0.25945946]
Covariance matrix setosa:
[[0.1233033  0.09071321 0.01573574 0.00939189]
 [0.09071321 0.14007508 0.01016517 0.00701201]
 [0.01573574 0.01016517 0.03836336 0.00781532]
 [0.00939189 0.00701201 0.00781532 0.01247748]]

Class: virginica
Mean vector virginica:
[6.57317073 2.98536585 5.52195122 2.02926829]
Covariance matrix virginica:
[[0.3705122  0.08409756 0.29035366 0.03655488]
 [0.08409756 0.10628049 0.07082927 0.03968902]
 [0.29035366 0.07082927 0.3122561  0.05034146]
 [0.03655488 0.03968902 0.05034146 0.06812195]]
```

Part d i

```
X_train = training_set[features].values
y_train = training_set[target].values
X_test = dev_set[features].values
y_test = dev_set[target].values

class_priors = {}
for class_index in np.unique(y_train):
    class_priors[class_index] = np.mean(y_train == class_index)

class_stats = {}
for class_index in np.unique(y_train):
    class_data = X_train[y_train == class_index]
    mean_vector = np.mean(class_data, axis=0)
    covariance_matrix = np.cov(class_data, rowvar=False)

    class_stats[class_index] = {
        'mean': mean_vector,
        'covariance': covariance_matrix
    }
```

```
def predict_class(X):
    num_classes = len(class_priors)
    num_samples = X.shape[0]
    posteriors = np.zeros((num_samples, num_classes))

    for class_index in range(num_classes):
        prior = class_priors[class_index]
        mean_vector = class_stats[class_index]['mean']
        covariance_matrix = class_stats[class_index]['covariance']

        #compute likelihood
        likelihood = multivariate_normal(mean=mean_vector, cov=covariance_matrix).pdf(X)

        #compute posterior probability
        posteriors[:, class_index] = likelihood * prior

    #predict the class with highest posterior probability
    predictions = np.argmax(posteriors, axis=1)
    return predictions

#predict the classes for the test set
y_pred = predict_class(X_test)

#print the predictions and labels
print("Predicted classes:", y_pred)
print("Actual classes:", y_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

```
Predicted classes: [0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2]
Actual classes: [0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2]
Accuracy: 1.00
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        13
  versicolor       1.00      1.00      1.00         8
   virginica       1.00      1.00      1.00         9

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

Part dii

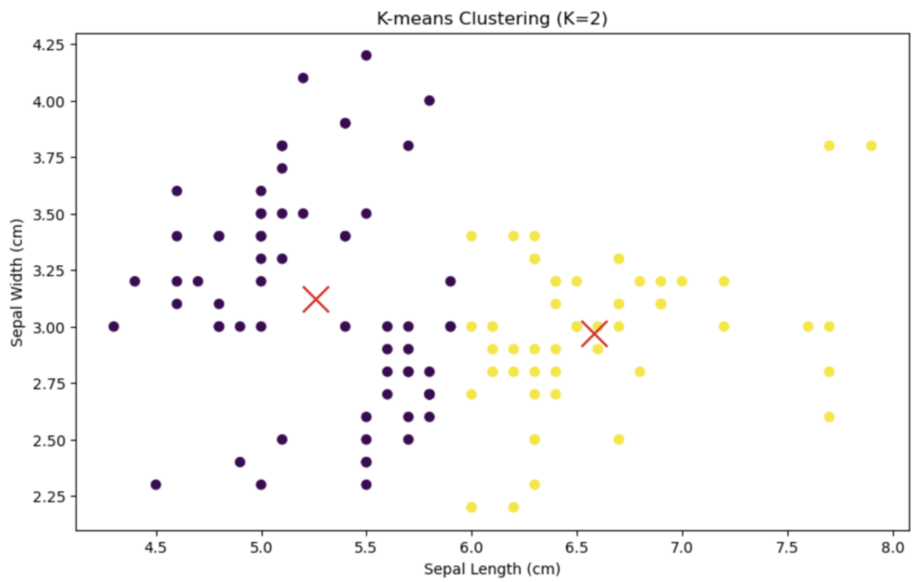
I will be using accuracy because I want to measure the performance of the classifier, as opposed to F1 that is about imbalance or precision which is concerned about false positives/negatives. The accuracy of the algorithm is 1.00 which is usually strange but makes sense in this context since the data set is extremely small and very well filtered.

Part e

```python
kmeans = KMeans(n_clusters=2, random_state=42)
training_set['cluster'] = kmeans.fit_predict(training_set[[feature_1, feature_2]])

plt.figure(figsize=(10, 6))

plt.figure(figsize=(10, 6))
plt.scatter(training_set[feature_1], training_set[feature_2], c=training_set['cluster'], cmap='viridis', marker='o')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red', marker='x')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.title('K-means Clustering (K=2)')
plt.show()
```



I decided to try and run the clusters and it does seem to change, setosa in particular seems to take over the points where versicolor used to be. However, the virginica cluster is pretty similar to the one in part a, plus a few points from versicolor that were overlapping.

Written Exercises

a. Naïve Bayes with binary features

i. The Naïve Bayes assumption in this context is that bikes and skis are conditionally independent when given a student's program (Master's or phd)

ii. The probability of a student in this group who neither bikes or skis being a master student is:
Prior probabilities:
P(y=0)=20/50=0.4 and P(y=1)=30/50=0.6
Likelihoods:
P(x1 = 0, x2 = 0 | y = 0) =
P(x1 = 0 | y = 0) x P(x2 = 0 | y = 0) =
0.75 x0.75 = 0.5625 P(x1 = 0, x2 = 0 | y = 1) =
P(x1 = 0 | y = 1) x P(x2 = 0 | y = 1) =
0.333 x 0.5 = 0.16665

P(x1 = 0, x2 = 0) = P(x1 = 0, x2 = 0 | y = 0) x P(y = 0) + P(x1 = 0, x2 = 0 | y = 1) x P(y = 1)P(x1 = 0, x2 = 0) =
0.5625 x 0.4 + 0.16665 x 0.6 P(x1 = 0, x2 = 0) =
0.225 + 0.09999 = 0.32499

Posterior Probability:

P(y = 0 | x1 = 0, x2 = 0) =
(0.5625*0.4)/(0.32499)P(y = 0 | x1 = 0, x2 = 0) =
0.225/0.32499 P(y = 0 | x1 = 0, x2 = 0) = 0.6924

iii. It would not make sense to assume that the probability of biking and skiing are conditionally independent for a phd student. I would change my answer in part b by stating that if a student cannot bike, they cannot ski either and vice versa.

b. Categorical Naïve Bayes

i.

$D = \{(x^i, y^i) \mid y \in \{1, 2, \ldots, k\}\}$

$P_\theta(y = k) = \phi_k$

Likelihood Function:

$L(\theta; D) = \prod_{i=1}^{n} P_\theta(x^i, y^i)$

Joint Probability:

$P_\theta(x^i, y^i) = P_\theta(y^i) P_\theta(x^i \mid y^i)$

$P_\theta(x^i \mid y^i = k) = \prod_{j=1}^{d} P_\theta(x_j^i \mid y^i = k) = \prod_{j=1}^{d} \psi_{jkx_j^i}$

$P_\theta(x^i, y^i) = \phi_{y^i} \prod_{j=1}^{d} \psi_{jy^ix_j^i}$

Log-Likelihood Function:

$\log L(\theta; D) = \sum_{i=1}^{n} \log P_\theta(x^i, y^i) \log L(\theta; D) = \sum_{i=1}^{n} (\log \phi_{y^i} + \sum_{j=1}^{d} \log \psi_{jy^ix_j^i})$

$\log L(\theta; D) = \sum_{i=1}^{n} \log \phi_{y^i} + \sum_{i=1}^{n} \sum_{j=1}^{d} \log \psi_{jy^ix_j^i}$

$= \sum_{k=1}^{k} \sum_{j=1}^{d} \sum_{l=1}^{k} \log P(x_j^i \mid y^i; \psi_{jkl}) + \sum_{i=1}^{n} \log P(y^i; \phi_k)$

$$\max_{\phi} \sum_{i=1}^{n} \log P_\theta(y = y^i; \phi)$$

$$= \sum_{i=1}^{n} \log \left(\frac{\phi_{y^i}}{\sum_{k=1}^{k} \phi_k} \right) = \sum_{i=1}^{n} \log \phi_{y^i} - n \log \sum_{k=1}^{k} \phi_k$$

$$= \sum_{k=1}^{k} \sum_{i:y^i=k} \log \phi_k - n \log \sum_{k=1}^{k} \phi_k$$

Derivatives

$$\frac{\phi_k}{\sum_{m=1}^{k} \phi_m} = \frac{n_k}{n}$$

Because $\Sigma_{m=1}^{k} \Phi_m = 1$ then
$\boxed{\Phi_k^* = \frac{n_k}{n}}$

ii.

objective $J(\hat{\Psi})$ is max $\Sigma_{k=1}^{k} \Sigma_{j=1}^{d} \Sigma_{\ell=1}^{L} \log \mathbb{P}(X_j^i | y^i; \Psi_{jk\ell})$

$$\Sigma_{k=1}^{k} \Sigma_{j=1}^{d} \Sigma_{\ell=1}^{L} \log \mathbb{P}(X_j^i | y^i; \Psi_{jk\ell}) = \sum_{k=1}^{k} \sum_{j=1}^{d} \sum_{\ell=1}^{L} n_{jk\ell} \log \Psi_{jk\ell}$$

our constraints are $\Psi_{jk\ell} \geq 0$ and $\Sigma_{\ell=1}^{L} \Psi_{jk\ell} = 1$.

Using lagrangian multiplier
$$\sum_{k=1}^{k} \sum_{j=1}^{d} \sum_{\ell=1}^{L} n_{jk\ell} \log \Psi_{jk\ell} = \sum_{k=1}^{k} \sum_{j=1}^{d} \sum_{\ell=1}^{L} n_{jk\ell} \log \Psi_{jk\ell} - \lambda \left(\sum_{jk\ell} \Psi_{jk\ell} - 1 \right)$$

Derivatives
$$\frac{n_{jk\ell}}{\Psi_{jk\ell}} - \lambda = 0 \Rightarrow \boxed{\Psi_{jk\ell} = \frac{n_{jk\ell}}{\lambda}}$$

c. Weights for Clustering

$$d_e^{(w)}(x_i, x_{i'}) = \frac{\Sigma_{\ell=1}^{p} w_\ell (x_{i\ell} - x_{i'\ell})^2}{\Sigma_{\ell=1}^{p} w_\ell}$$

satisfies $d_e^{(w)}(x_i, x_{i'}) = d_e(z_i, z_{i'}) = \Sigma_{\ell=1}^{p} (z_{i\ell} - z_{i'\ell})^2$

where $z_{i\ell} = x_{i\ell} \cdot \left(\frac{w_\ell}{\Sigma_{\ell=1}^{p} w_\ell} \right)^{1/2}$

Compute distance
$$d_e(z_i, z_{i'}) = \left(\sum_{\ell=1}^{p} (z_{i\ell} - z_{i'\ell})^2 \right)^{1/2}$$

substitute with z_{il}

$$d_e(z_i, z_i') = \left(\sum_{l=1}^{p} \left(x_{il} \cdot \left(\frac{w_l}{\sum_{l=1}^{p} w_l}\right)^{1/2} - x_{il}' \cdot \left(\frac{w_l}{\sum_{l=1}^{p} w_l}\right)^{1/2}\right)^2\right)^{1/2}$$

Simplify

$$d_e(z_i, z_i') = \left(\sum_{l=1}^{p} \left(\left(\frac{w_l}{\sum_{l=1}^{p} w_l}\right)^{1/2} (x_{il} - x_{il}')^2\right)^{1/2}$$

$$= \left(\sum_{l=1}^{p} \frac{w_l}{\sum_{l=1}^{p} w_l} (x_{il} - x_{il}')^2\right)^{1/2}$$

$$= \left(\sum_{l=1}^{p} \frac{w_l (x_{il} - x_{il}')^2}{\sum_{l=1}^{p} w_l}\right)^{1/2}$$

so

$$\left(\sum_{l=1}^{p} \frac{w_l (x_{il} - x_{il}')^2}{\sum_{l=1}^{p} w_l}\right)^{1/2} = \left(\sum_{l=1}^{p} \frac{w_l (x_{il} - x_{il}')^2}{\sum_{l=1}^{p} w_l}\right)^{1/2}$$

and $\boxed{d_e^{(w)}(x_i, x_{i_j}') = d_e(z_i, z_i')}$