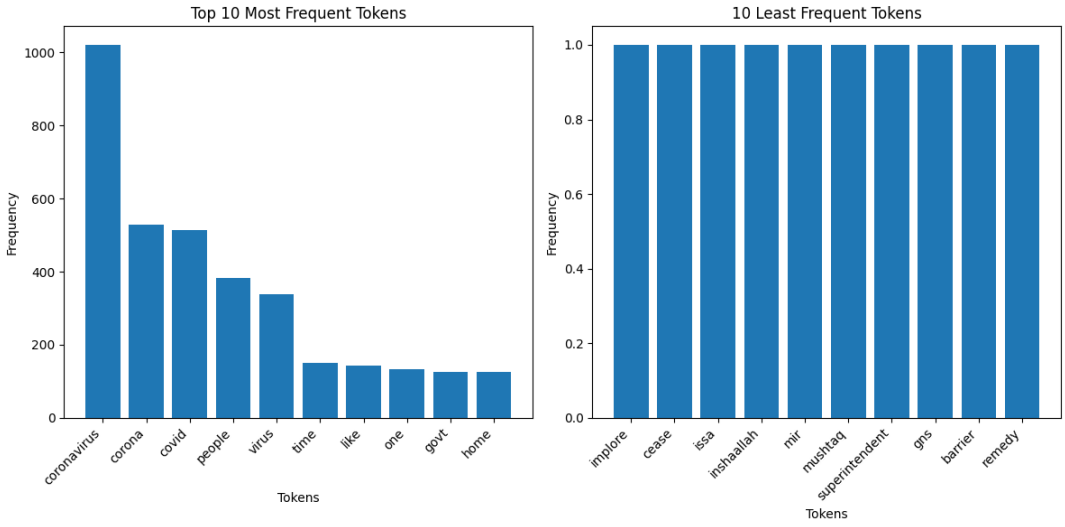
گزارش تمرین ۳ یادگیری ماشین

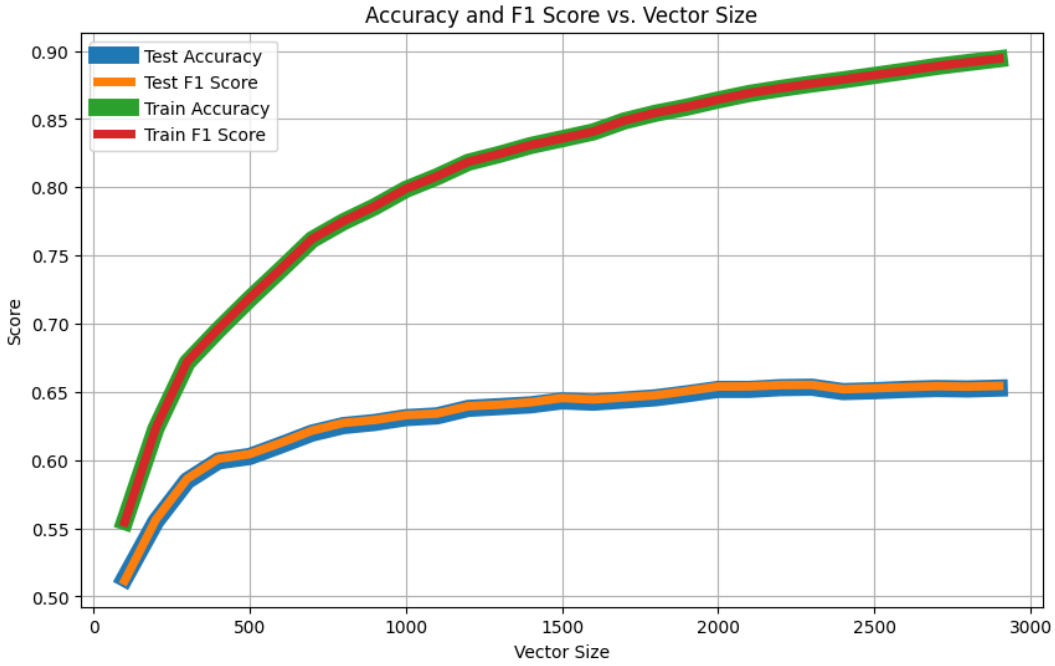
مازندرانیان

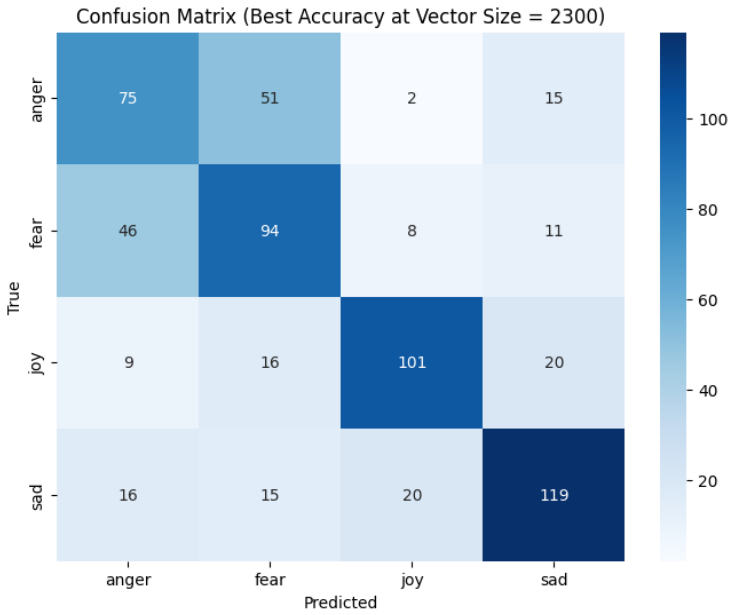
۸۳۰۴۰۲۰۶۶

سوال ۱)

ابندا طبق خواسته های سوال پیاده سازی انجام شد ولی در بهترین حالت و برای vector_size=3000 دقتی برابر با ۶۵ درصد استخراج شد. همچنین با کتابخانه ی sklearn و روش TFIDF که معمولا روش بهتری نسبت به Bag Of Words است هم به دقت بیشتری دست پیدا نکردیم. عمده مشکل این دقت پایین برای کلاس های anger و fear بود که مدل پیش بینی های این کلاس ها رو اکثرا جابجا انجام میداد. به عنوان روش جایگزین کلماتی را که بین ۴ کلاس برای هر کدام متمایز هستند را در متن نگه میداریم و مابقی کلمات را حذف میکنیم. همچنین با استفاده از کتابخانه ی nltk کلمات stop words هم حذف شدند. این کار منجر به این شد که دقت مدل ۱۰۰ درصد شود که البته نشان دهنده ی overfit بودن روی این دیتاست است.







Naive Bayes Python

```
def simple_bayes_classifier(X_train, y_train, X_test, vocab):
    class_counts = {}
    word_counts = {}

    for i in range(len(X_train)):
        label = y_train[i]
        if label not in class_counts:
            class_counts[label] = 0
            word_counts[label] = defaultdict(int)
        class_counts[label] += 1
        for word in X_train[i]:
            word_counts[label][word] += 1

    predictions = []
    for document in X_test:
        probs = {}
        for label in class_counts:
            prob = np.log(class_counts[label] / len(X_train))
            for word in document:
                if word in vocab:
                  prob += np.log((word_counts[label][word] + 1) /
(sum(word_counts[label].values()) + len(vocab)))
            probs[label] = prob


            fear_count = sum(1 for word in document if word in [word for word, count in
fear_top_words])
            anger_count = sum(1 for word in document if word in [word for word, count in
anger_top_words])

        if anger_count > fear_count:
            probs['anger'] /= 2
        elif fear_count > anger_count:
            probs['fear'] /= 2


        predictions.append(max(probs, key=probs.get))
    return predictions
```

این تابع یک مدل ساده‌ی بیزین برای دسته‌بندی متون پیاده‌سازی کرده است. هدف آن پیش‌بینی برچسب مناسب برای هر سند جدید بر اساس داده‌های آموزشی است. ابتدا داده‌های آموزشی را بررسی کرده و تعداد اسناد مربوط به هر برچسب (کلاس) و تعداد تکرار کلمات در هر برچسب را می‌شمارد. این اطلاعات برای محاسبه احتمال وقوع هر کلاس و احتمال شرطی کلمات در آن کلاس استفاده می‌شود.

سپس برای اسناد جدید، احتمال تعلق به هر کلاس با استفاده از احتمالات محاسبه‌شده قبلی تخمین زده می‌شود. در این فرآیند، اگر کلمات مربوط به احساسات خاصی (مثل ترس یا خشم) در متن وجود داشته باشند، احتمال کلاس مرتبط با آن احساس کاهش می‌یابد. در نهایت، کلاسی که بیشترین احتمال را دارد به عنوان پیش‌بینی برچسب متن انتخاب می‌شود.

$$P(w|C) = \frac{\text{تعداد کلمه } w \text{ در کلاس } C + 1}{\text{مجموع تمام کلمات در کلاس } C + |\text{واژگان}|}$$

سوال ۲)

```
max_parents = 1:
train accuracy: 0.9447
train F1-Score: 0.4858
test accuracy: 0.9503
test F1-Score: 0.4873
max_parents = 2:
train accuracy: 0.9993
train F1-Score: 0.9966
test accuracy: 0.9997
test F1-Score: 0.9982
max_parents = 3:
train accuracy: 0.9993
train F1-Score: 0.9966
test accuracy: 0.9997
test F1-Score: 0.9982
max_parents = 4:
train accuracy: 0.9993
train F1-Score: 0.9966
test accuracy: 0.9997
test F1-Score: 0.9982
max_parents = 5:
train accuracy: 0.9994
train F1-Score: 0.9973
test accuracy: 0.9983
test F1-Score: 0.9913
```

K2 Python

```
max_parents = 1:
asia: []
tub: ['asia']
smoke: ['tub']
bronc: ['smoke']
either: ['tub']
xray: ['either']
dysp: ['bronc']
lung: ['either']
max_parents = 2:
asia: []
tub: ['asia']
smoke: ['tub', 'asia']
bronc: ['smoke', 'tub']
either: ['tub', 'smoke']
xray: ['either', 'tub']
dysp: ['bronc', 'either']
lung: ['either', 'tub']
max_parents = 3:
asia: []
tub: ['asia']
smoke: ['tub', 'asia']
bronc: ['smoke', 'tub', 'asia']
either: ['tub', 'smoke', 'asia']
xray: ['either', 'tub', 'bronc']
dysp: ['bronc', 'either', 'asia']
lung: ['either', 'tub', 'smoke']
max_parents = 4:
asia: []
tub: ['asia']
smoke: ['tub', 'asia']
bronc: ['smoke', 'tub', 'asia']
either: ['tub', 'smoke', 'asia', 'bronc']
xray: ['either', 'tub', 'bronc', 'smoke']
dysp: ['bronc', 'either', 'asia', 'xray']
lung: ['either', 'tub', 'smoke', 'bronc']
max_parents = 5:
asia: []
tub: ['asia']
smoke: ['tub', 'asia']
bronc: ['smoke', 'tub', 'asia']
either: ['tub', 'smoke', 'asia', 'bronc']
xray: ['either', 'tub', 'bronc', 'smoke', 'asia']
dysp: ['bronc', 'either', 'asia', 'xray', 'smoke']
lung: ['either', 'tub', 'smoke', 'bronc', 'dysp']
```

K2 Python

```python
def log_likelihood_score(variable, parents, data):
    if not parents:
        counts = data[variable].value_counts().values
        likelihood = np.sum(counts * np.log(counts + 1e-10)) - len(data) * np.log(len(data))
        return likelihood

    parent_counts = data.groupby(parents)[variable].value_counts().unstack().fillna(0)
    likelihood = 0
    for row in parent_counts.values:
        total = np.sum(row)
        if total > 0:
            likelihood += np.sum(row * np.log(row + 1e-10)) - total * np.log(total + 1e-10)
    return likelihood

def K2_algorithm(data, max_parents):
    n_variables = len(data.columns)
    parents = {var: [] for var in data.columns}

    for i in range(1, n_variables):
        current_var = data.columns[i]
        current_score = log_likelihood_score(current_var, [], data)

        while True:
            possible_parents = [p for p in data.columns[:i] if p not in parents[current_var]]
            if len(possible_parents) == 0 or len(parents[current_var]) >= max_parents:
                break

            best_parent = None
            best_score = current_score

            for parent in possible_parents:
                new_parents = parents[current_var] + [parent]
                new_score = log_likelihood_score(current_var, new_parents, data)
                if new_score > best_score:
                    best_score = new_score
                    best_parent = parent

            if best_parent is not None:
                parents[current_var].append(best_parent)
                current_score = best_score
            else:
                break

    return parents
```

در ابتدا، برای هر متغیر یک امتیاز اولیه محاسبه می‌شود که نشان می‌دهد آن متغیر بدون والدین چه قدر خوب مدل‌سازی می‌شود. سپس الگوریتم شروع به جست‌وجوی بهترین والدین برای هر متغیر می‌کند. والدین یک متغیر، متغیرهایی هستند که تأثیر

مستقیم روی آن دارند. برای پیدا کردن والدین، الگوریتم والدینی را اضافه می‌کند که بیشترین بهبود را در امتیاز متغیر ایجاد کنند. این کار ادامه پیدا می‌کند تا وقتی که یا هیچ بهبودی در امتیاز حاصل نشود یا تعداد والدین از مقدار مجاز بیشتر نشود.

در نهایت، خروجی این الگوریتم یک ساختار شبکه است که برای هر متغیر، لیستی از والدینش را نشان می‌دهد. این ساختار می‌تواند در مسائل مختلفی مثل پیش‌بینی و تحلیل روابط بین متغیرها استفاده شود.