МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)
ИНСТИТУТ «ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И ПРИКЛАДНАЯ МАТЕМАТИКА»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Искусственный интеллект»

ЛАБОРАТОРНАЯ РАБОТА №1

VI семестр

Студент: Калинина А.В.
Группа: М8О-308Б-19
Преподаватель: Самир Ахмед

Подпись: _____
Оценка: _____
Дата сдачи: «___»_____22г.
Дата проверки: «___»_____22г.

Москва, 2022

1. Постановка задачи

1) реализовать следующие алгоритмы машинного обучения: Linear/ Logistic Regression, SVM, KNN, Naive Bayes в отдельных классах
2) Данные классы должны наследоваться от BaseEstimator и ClassifierMixin, иметь методы fit и predict
3) Вы должны организовать весь процесс предобработки, обучения и тестирования с помощью Pipeline
.4) Вы должны настроить гиперпараметры моделей с помощью кросс валидации.
5)Проделать аналогично с коробочными решениями
6) Для каждой модели получить оценки метрик:Confusion Matrix, Accuracy, Recall, Precision, ROC_AUC curve
7) Проанализировать полученные результаты и сделать выводы о применимости моделей

2. Подготовка данных

Для выполнения лабораторной работы в лабораторной работе №0 был произведен предварительный анализ и подготовка данных. А именно, произведена нормализация признаков и корректировка баланса классов. Для разбиения данных на обучающую и тестовую выборки используем train_test_split из библиотеки sklearn.

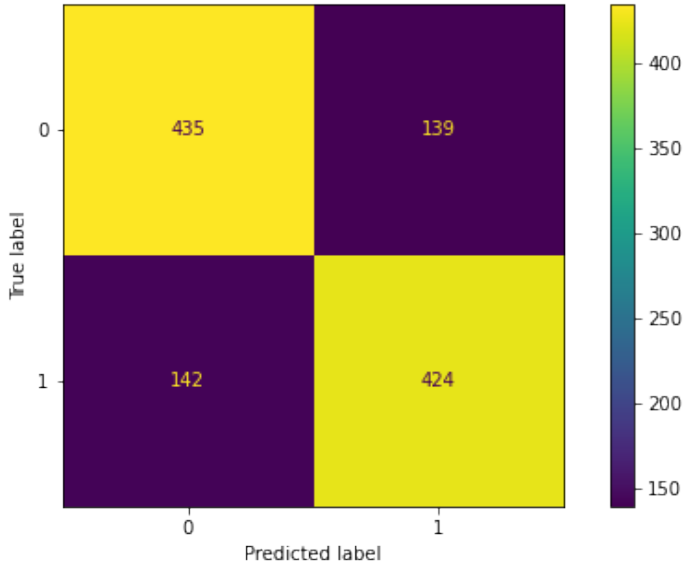
3. SVM

```
class SVM(BaseEstimator, ClassifierMixin):
    def __init__(self, lr=0.1, batch=10, epochs=1, alpha=0.0001):
        self.lr = lr
        self.batch = batch
        self.epochs = epochs
        self.alpha = alpha
    def fit(self, data, labels):
        self.w = np.random.normal(0, 1, (data.shape[1]+1,))
             data  =  np.concatenate((data,  np.ones((data.shape[0],1))),
axis=1)
        labels = labels * 2 - 1
        for _ in range(self.epochs):
            for i in range(self.batch, len(data), self.batch):
                data_batch = data[i-self.batch:i]
                labels_batch = labels[i-self.batch:i]

                grad = 2 * self.alpha * self.w
                for i, x in enumerate(data_batch):
                    if 1 - x.dot(self.w) * labels_batch[i] > 0:
                        grad -= x * labels_batch[i]

                self.w -= self.lr * grad
        return self
```

```
    def predict(self, data):
                                    return    (np.sign(np.concatenate((data,
np.ones((data.shape[0],1))), axis=1).dot(self.w)) + 1) / 2
```

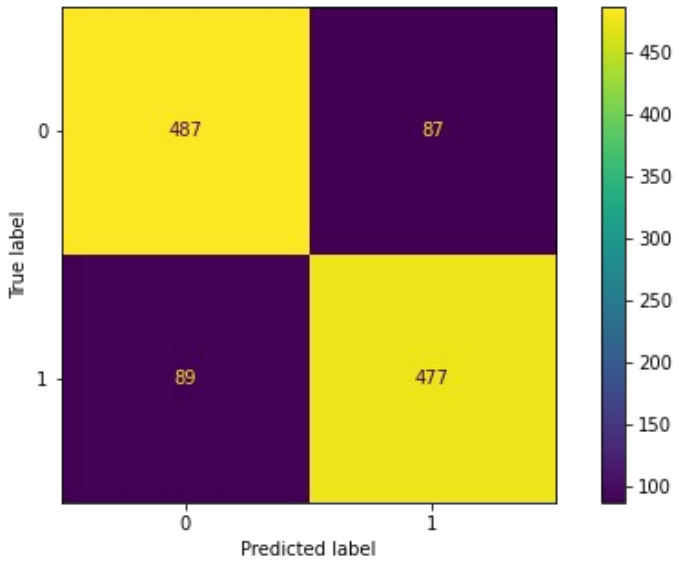
Результат работы

```
Accuracy: 0.7535087719298246
Precision: 0.7531083481349912
Recall: 0.7491166077738516
```



Готовый классификатор

```
Accuracy: 0.8456140350877193
Precision: 0.8457446808510638
Recall: 0.842756183745583
```

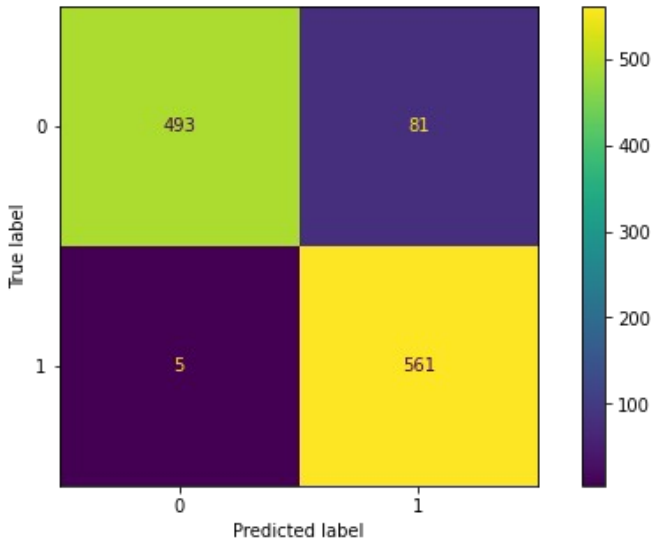


4. KNN

```
class KNN(BaseEstimator, ClassifierMixin):
    def __init__(self, k = 1):
        self.k = k

    def fit(self, data, labels):
        self.data = data
        self.labels = labels
        return self

    def predict(self, data):
        res = np.ndarray((data.shape[0],))
        for i, x in enumerate(data):
            distances = euclidean_distances([x], self.data)[0]
            neighbors = np.argpartition(distances, kth = self.k - 1)
            k_neighbors = neighbors[:self.k]
            values, counts = np.unique(self.labels[k_neighbors],
return_counts = True)
            res[i] = values[counts.argmax()]
        return res
```

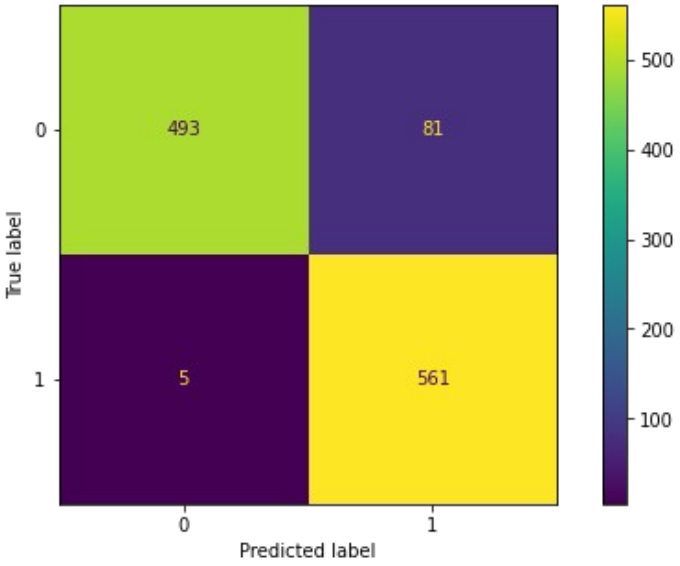
Результат работы

```
Accuracy: 0.9245614035087719

Precision: 0.8738317757009346
Recall: 0.991166077738516
```



Готовый классификатор

```
Accuracy: 0.9245614035087719

Precision: 0.8738317757009346
Recall: 0.991166077738516
```

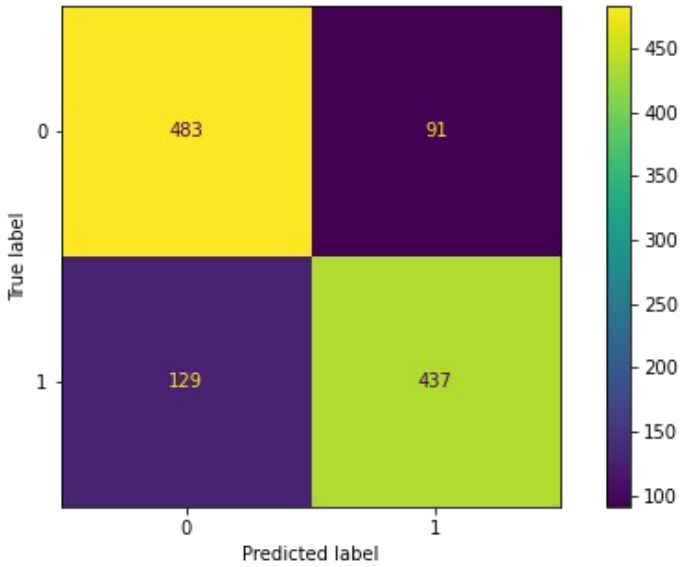


5. Naive Bayes

```
class NaiveBayes(BaseEstimator, ClassifierMixin):
    def __init__(self):
        pass

    def normal_gauss(self, x, mu, sigma):
        return (np.exp(-((x - mu) / sigma)**2 / 2)) / np.float32(sigma
* np.sqrt(2 * np.pi))

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        self.X = X
        self.y = y
        labels, counts = np.unique(self.y, return_counts=True)
        self.standard_deviations = np.array([self.X[self.y ==
label].std(axis=0) for label in labels])
        self.means = np.array([self.X[self.y == label].mean(axis=0) for
label in labels])
        self.y_pred = np.array([count / self.y.shape[0] for count in
counts])
        self.labels = labels
        return self

    def predict(self, X):
        check_is_fitted(self, ['X', 'y'])
        result = np.ndarray(X.shape[0])
        for (num_x, x) in enumerate(X):
            predictions = np.array(self.y_pred)
            for (num_label, label) in enumerate(self.labels):
                predictions[num_label] *=
np.prod(np.array([self.normal_gauss(x[i], self.means[num_label][i],
self.standard_deviations[num_label][i]) for i in range(X.shape[1])]))
            result[num_x] = np.argmax(predictions)
        return result
```

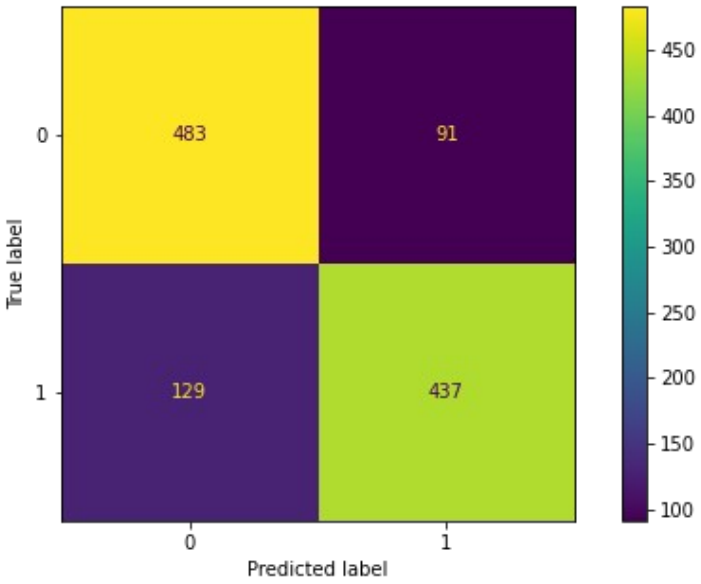
Результат работы

```
Accuracy: 0.8070175438596491
Precision: 0.8276515151515151
Recall: 0.7720848056537103
```



Готовый классификатор

```
Accuracy: 0.8070175438596491
Precision: 0.8276515151515151
Recall: 0.7720848056537103
```



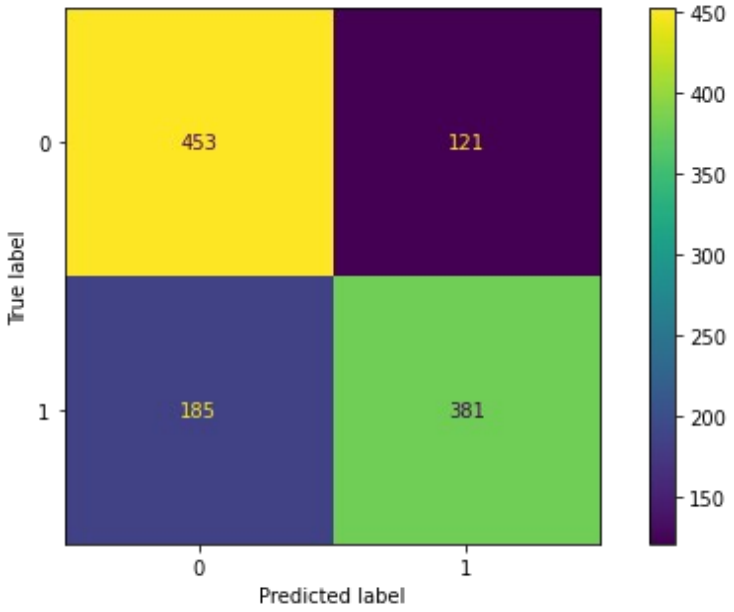
6. Logistic Regression

```
class LogisticRegression(BaseEstimator, ClassifierMixin):
    def __init__(self, lr=0.1, batch=10, epochs=1, alpha=0.0001):
        self.lr = lr
        self.batch = batch
        self.epochs = epochs
```

```
        self.alpha = alpha

    def fit(self, data, labels):
        self.w = np.random.normal(0, 1, (data.shape[1]+1,))
        data = np.concatenate((data, np.ones((data.shape[0],1))),
axis=1)
        for _ in range(self.epochs):
            for i in range(self.batch, len(data), self.batch):
                data_batch = data[i-self.batch:i]
                labels_batch = labels[i-self.batch:i]

                pred = self.sigmoid(np.dot(self.w, data_batch.T))
                grad = 2 * self.alpha * self.w + np.dot(pred -
labels_batch, data_batch)

                self.w -= self.lr * grad
        return self

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def predict(self, data):
        return (self.sigmoid(np.concatenate((data,
np.ones((data.shape[0],1))), axis=1).dot(self.w)) >
0.5).astype('int64')
```

Результат работы

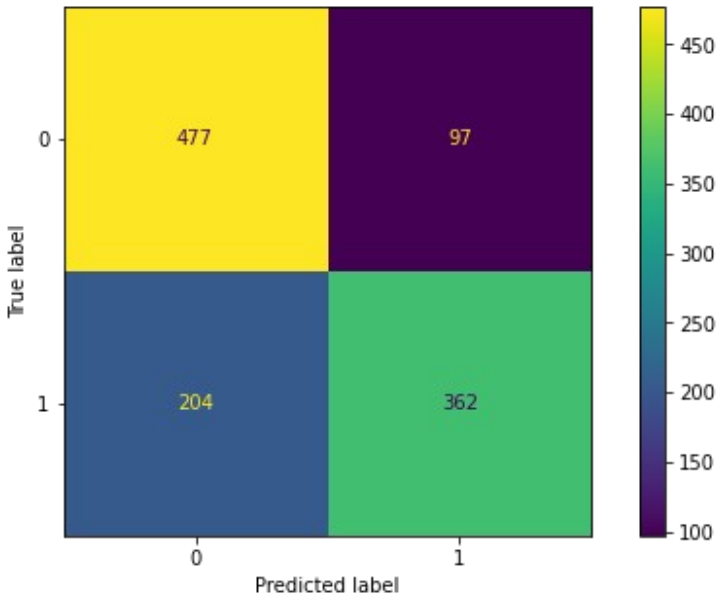
```
Accuracy: 0.7315789473684211
Precision: 0.7589641434262948
Recall: 0.6731448763250883
```



Готовый классификатор

```
Accuracy: 0.7359649122807017
```

```
Precision: 0.7886710239651417
Recall: 0.6395759717314488
```



7. Выводы

В ходе выполнения лабораторной работы были изучены модели классического машинного обучения. А именно: логистическая регрессия, наивный байесовский классификатор, метод опорных векторов и метод К ближайших соседей. Все модели показали хорошие результаты классификации для представленного набора с данными.