Московский авиационный институт
(национальный исследовательский университет)

Факультет: «Компьютерные науки и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа №1

по курсу «Машинное обучение»
Тема: «Линейные модели»

Студент: Мариничев И. А.
Группа: М8О-308Б-19
Преподаватель: Ахмед С. Х.
Оценка:

Москва
2022

1. Постановка задачи.

Вы собрали данные и их проанализировали, визуализировали и представили отчет своим партнерам и спонсорам. Они согласились, что ваша задача имеет перспективу и продемонстрировали заинтересованность в вашем проекте. Самое время реализовать прототип! Вы считаете, что нейронные сети переоценены (просто боитесь признаться, что у вас не хватает ресурсов и данных), и считаете, что за классическим машинным обучением будущее и потому собираетесь использовать классические модели. Вашим первым предположением является предположение, что данные и все в этом мире имеет линейную зависимость, ведь не зря же в конце каждой нейронной сети есть линейный слой классификации. В качестве первых моделей вы выбрали линейную/логистическую регрессию и SVM. Так как вы очень осторожны и боитесь ошибиться, вы хотите реализовать случай, когда все-таки мы не делаем никаких предположений о данных и взяли за основу идею "близкие объекты дают близкий ответ" и идею, что теорема Байеса имеет ранг королевской теоремы. Так как вы не доверяете другим людям, вы хотите реализовать алгоритмы сами с нуля без использования scikit-learn (почти). Вы хотите узнать, насколько хорошо ваши модели работают на выбранных вам данных и хотите замерить метрики качества. Ведь вам нужно еще отчитаться спонсорам! Формально говоря вам предстоит сделать следующее:

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, вывести и сохранить эти гиперпараметры в файл, вместе с обученными моделями;
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;
8. Загрузить полученные гиперпараметры модели и обученные модели в формате pickle на гит вместе с Jupyter Notebook ваших экспериментов.

2. Реализация алгоритмов машинного обучения.

На этапе обучения мы сравним мои реализации с моделями из scikit-learn. Ниже приведены реализации следующих алгоритмов:

- Linear/Logistic Regression
- SVM
- KNN
- Naive Bayes

Логистическая регрессия (англ. logistic regression)

Метод построения линейного классификатора, позволяющий оценивать апостериорные вероятности принадлежности объектов классам

```
from sklearn.preprocessing import OneHotEncoder
from scipy.special import softmax

class MyMulticlassLogisticRegression(ClassifierMixin, BaseEstimator):
    def __init__(self, max_iter=1000, lr=0.1, mu=0.0001):
        self.max_iter = max_iter
        self.lr = lr
        self.mu = mu

    def fit(self, X, y):

        # Check that X and y have correct shape
        X, y = check_X_y(X, y)

        self.X_ = X

        self._label_encoder = OneHotEncoder(sparse=False)
        self.y_ = self._label_encoder.fit_transform(y.reshape(-1, 1))

        self.w_ = np.zeros((X.shape[1], self.y_.shape[1]))

        # Gradient descent
        step = 0
        while step < self.max_iter:
            step += 1
            self.w_ -= self.lr * self._gradient()

            # Loss calculation
            Z = - X @ self.w_
            N = X.shape[0]
            self.loss = 1 / N * (np.trace(X @ self.w_ @ self.y_.T) +
np.sum(np.log(np.sum(np.exp(Z), axis = 1))))

        # Return the classifier
        return self

    def predict(self, X):

        # Check is fit had been called
        check_is_fitted(self, ['X_', 'y_'])

        # Input validation
        X = check_array(X)
```

```
        Z = -X @ self.w_
        P = softmax(Z, axis=1)
        return np.argmax(P, axis = 1) + 1

    def _gradient(self):
        Z = - self.X_ @ self.w_
        P = softmax(Z, axis=1)
        N = self.X_.shape[0]
        grad = 1 / N * (self.X_.T @ (self.y_ - P)) + 2 * self.mu * self.w_
        return grad
```

Метод опорных векторов (англ. SVM — support vector machine)

Основная идея метода заключается в построении гиперплоскости, разделяющей объекты выборки оптимальным способом. Алгоритм работает в предположении, что чем больше расстояние (зазор) между разделяющей гиперплоскостью и объектами разделяемых классов, тем меньше будет средняя ошибка классификатора

```
from sklearn.preprocessing import LabelEncoder

class MySVM(ClassifierMixin, BaseEstimator):
    def __init__(self, C=1, max_iter=50):
        self.C = C
        self.max_iter = max_iter

    def fit(self, X, y):

        # Check that X and y have correct shape
        X, y = check_X_y(X, y)

        self.X_ = X

        n_samples, n_features = X.shape

        # Normalize labels
        self._label_encoder = LabelEncoder()
        self.y_ = self._label_encoder.fit_transform(y)

        # Initialize primal and dual coefficients
        n_classes = len(self._label_encoder.classes_)
        self.dual_coef_ = np.zeros((n_classes, n_samples), dtype=np.float64)
        self.coef_ = np.zeros((n_classes, n_features))

        # Pre-compute norms
        norms = np.sqrt(np.sum(X ** 2, axis=1))

        ind = np.arange(n_samples)
        for it in range(self.max_iter):
            for j in range(n_samples):
                i = ind[j]

                # All-zero samples can be safely ignored
                if norms[i] == 0:
                    continue

                g = self._partial_gradient(i)

                # Solve subproblem for the ith sample
                delta = self.__solve_subproblem(g, norms, i)

                # Update primal and dual coefficients
```

```
                self.coef_ += (delta * X[i][:, np.newaxis]).T
                self.dual_coef_[:, i] += delta

        # Return the classifier
        return self

    def predict(self, X):

        # Check is fit had been called
        check_is_fitted(self, ['X_', 'y_'])

        # Input validation
        X = check_array(X)

        decision = np.dot(X, self.coef_.T)
        pred = decision.argmax(axis=1)
        return self._label_encoder.inverse_transform(pred)

    def _partial_gradient(self, i):
        # Partial gradient for the ith sample
        g = np.dot(self.X_[i], self.coef_.T) + 1
        g[self.y_[i]] -= 1
        return g

    def _projection_simplex(self, v, z=1):
        # Projection onto the simplex:
        #     w^* = argmin_w 0.5 ||w-v||^2 s.t. \sum_i w_i = z, w_i >= 0
        n_features = v.shape[0]
        u = np.sort(v)[::-1]
        cssv = np.cumsum(u) - z
        ind = np.arange(n_features) + 1
        cond = u - cssv / ind > 0
        rho = ind[cond][-1]
        theta = cssv[cond][-1] / float(rho)
        w = np.maximum(v - theta, 0)
        return w

    def __solve_subproblem(self, g, norms, i):
        # Prepare inputs to the projection.
        Ci = np.zeros(g.shape[0])
        Ci[self.y_[i]] = self.C
        beta_hat = norms[i] * (Ci - self.dual_coef_[:, i]) + g / norms[i]
        z = self.C * norms[i]

        # Compute projection onto the simplex.
        beta = self._projection_simplex(beta_hat, z)

        return Ci - self.dual_coef_[:, i] - beta / norms[i]
```

Метод k ближайших соседей (англ. KNN — k Nearest Neighbours)

Для повышения надёжности классификации объект относится к тому классу, которому принадлежит большинство из его соседей, то есть k ближайших к нему объектов обучающей выборки

```
class MyKNN(ClassifierMixin, BaseEstimator):
    def __init__(self, k=5):
        self.k = k

    def fit(self, X, y):
        # Check that X and y have correct shape
        X, y = check_X_y(X, y)
```

```
        self.X_ = X
        self.y_ = y
        # Return the classifier
        return self

    def predict(self, X):

        # Check is fit had been called
        check_is_fitted(self, ['X_', 'y_'])

        # Input validation
        X = check_array(X)

        pred = np.ndarray((X.shape[0],))
        for i, x in enumerate(X):
            neighbors = np.argpartition(((self.X_ - X[i]) ** 2).sum(axis=1),
self.k - 1)[:self.k]
            values, counts = np.unique(self.y_[neighbors], return_counts=True)
            pred[i] = values[counts.argmax()]
        return pred
```

Наивный байесовский классификатор (англ. naive Bayes)

Простой вероятностный классификатор, основанный на применении теоремы Байеса со строгими (наивными) предположениями о независимости

```
# Bayes Theorem form
# P(y|X) = P(X|y) * P(y) / P(X)
class MyNaiveBayes(ClassifierMixin, BaseEstimator):
    def fit(self, X, y):
        # Check that X and y have correct shape
        X, y = check_X_y(X, y)

        # Store the classes seen during fit
        self.classes_ = np.unique(y)

        self.X_ = X
        self.y_ = y

        self.count_ = len(self.classes_)
        self.rows_ = X.shape[0]

        self.mean_, self.var_ = self._calc_statistics()
        self.prior_ = self._calc_prior()

        # Return the classifier
        return self

    def predict(self, X):
        # Check is fit had been called
        check_is_fitted(self, ['X_', 'y_'])

        # Input validation
        X = check_array(X)

        pred = [self._calc_posterior(f) for f in X]
        return pred

    def _calc_prior(self):
        # Prior probability P(y)
        # Calculate prior probabilities
        prior = (pd.DataFrame(self.X_).groupby(self.y_).apply(lambda x: len(x))
/ self.rows_).to_numpy()
```

```
        return prior

    def _calc_statistics(self):
        # Calculate mean, variance for each column and convert to numpy array
        mean =
pd.DataFrame(self.X_).groupby(self.y_).apply(pd.DataFrame.mean).to_numpy()
        var =
pd.DataFrame(self.X_).groupby(self.y_).apply(pd.DataFrame.var).to_numpy()

        return mean, var

    def _gaussian_density(self, class_idx, x):
        # Calculate probability from gaussian density function (normally
distributed)
        # we will assume that probability of specific target value given
specific class is normally distributed

        # Probability density function derived from wikipedia:
        # (1/√2pi*σ) * exp((-1/2)*((x-μ)^2)/(2*σ²)), where μ is mean, σ² is
variance, σ is quare root of variance (standard deviation)
        mean = self.mean_[class_idx]
        var = self.var_[class_idx]
        numerator = np.exp((-1/2)*((x-mean)**2) / (2 * var))
        denominator = np.sqrt(2 * np.pi * var)
        prob = numerator / denominator
        return prob

    def _calc_posterior(self, x):
        posteriors = []

        # Calculate posterior probability for each class
        for i in range(self.count_):
            prior = np.log(self.prior_[i]) # use the log to make it more
numerically stable
            conditional = np.sum(np.log(self._gaussian_density(i, x))) # use the
log to make it more numerically stable
            posterior = prior + conditional
            posteriors.append(posterior)
        # Return class with highest posterior probability
        return self.classes_[np.argmax(posteriors)]
```

3. Обучение

Теперь нам необходимо провести настройку гиперпараметров для моделей, у которых таковые имеются.

Затем нам нужно обучить модели с лучшими параметрами и провести оценку метрик:

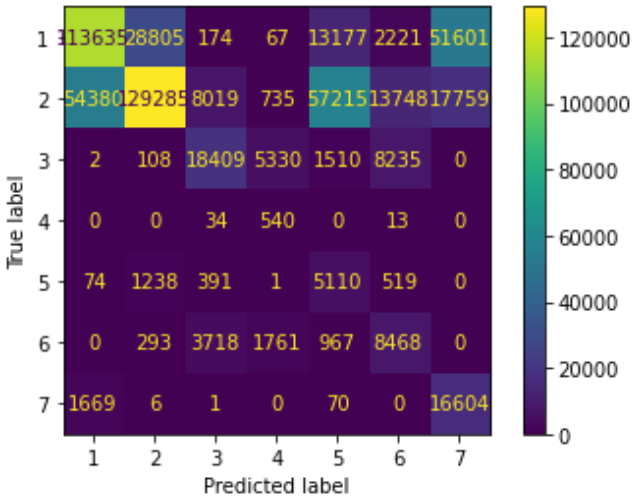
- Confusion Matrix ✓
- Accuracy ✓
- Recall ✓
- Precision ✓
- ROC_AUC curve ✗ (т.к. применяется лишь для бинарной классификации)

Теперь перейдем непосредственно к обучению и сравнению реализаций

Logistic Regression

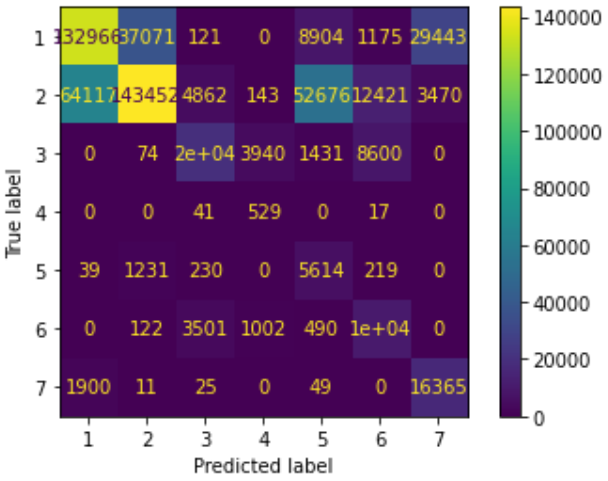
1. MyMulticlassLogisticRegression

	precision	recall	f1-score	support
Spruce/Fir (1)	0.67	0.54	0.60	209680
Lodgepole Pine (2)	0.81	0.46	0.59	281141
Ponderosa Pine (3)	0.60	0.55	0.57	33594
Cottonwood/Willow (4)	0.06	0.92	0.12	587
Aspen (5)	0.07	0.70	0.12	7333
Douglas-fir (6)	0.26	0.56	0.35	15207
Krummholz (7)	0.19	0.90	0.32	18350
accuracy			0.52	565892
macro avg	0.38	0.66	0.38	565892
weighted avg	0.70	0.52	0.57	565892



2. LogisticRegression

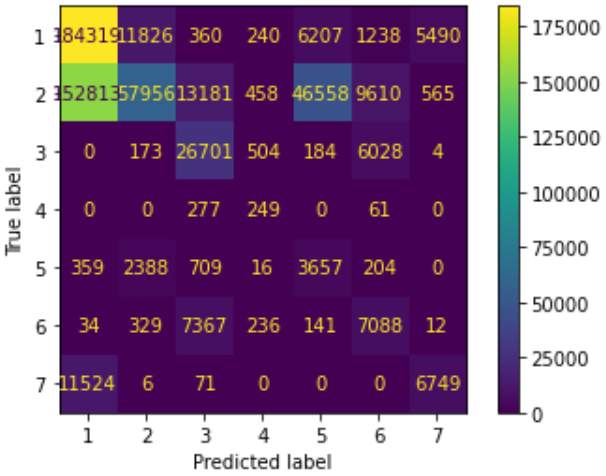
	precision	recall	f1-score	support
Spruce/Fir (1)	0.67	0.63	0.65	209680
Lodgepole Pine (2)	0.79	0.51	0.62	281141
Ponderosa Pine (3)	0.69	0.58	0.63	33594
Cottonwood/Willow (4)	0.09	0.90	0.17	587
Aspen (5)	0.08	0.77	0.15	7333
Douglas-fir (6)	0.31	0.66	0.42	15207
Krummholz (7)	0.33	0.89	0.48	18350
accuracy			0.58	565892
macro avg	0.42	0.71	0.45	565892
weighted avg	0.70	0.58	0.62	565892



SVM

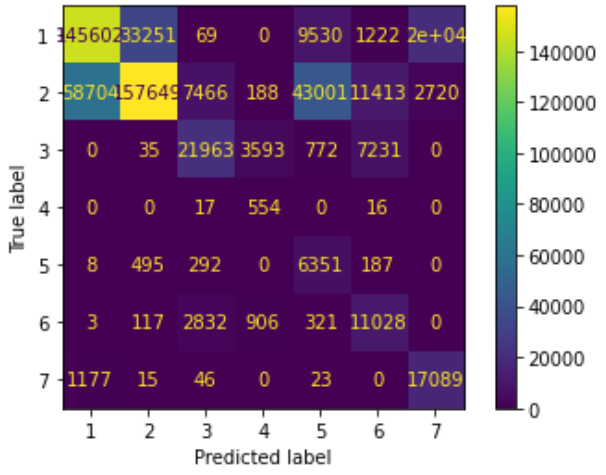
1. MySVM

	precision	recall	f1-score	support
Spruce/Fir (1)	0.53	0.88	0.66	209680
Lodgepole Pine (2)	0.80	0.21	0.33	281141
Ponderosa Pine (3)	0.55	0.79	0.65	33594
Cottonwood/Willow (4)	0.15	0.42	0.22	587
Aspen (5)	0.06	0.50	0.11	7333
Douglas-fir (6)	0.29	0.47	0.36	15207
Krummholz (7)	0.53	0.37	0.43	18350
accuracy			0.51	565892
macro avg	0.41	0.52	0.39	565892
weighted avg	0.65	0.51	0.47	565892



2. SVC

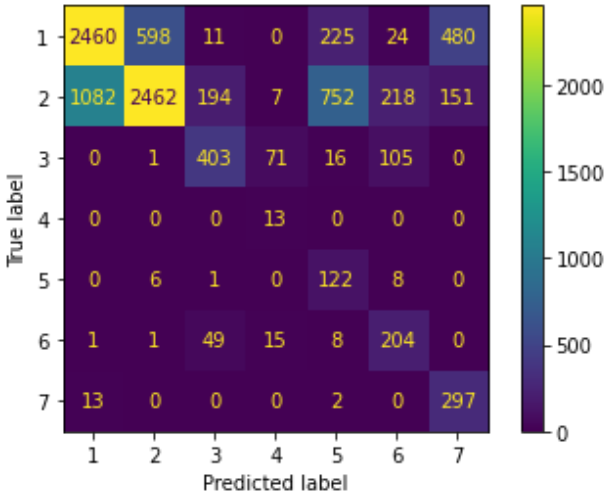
	precision	recall	f1-score	support
Spruce/Fir (1)	0.71	0.69	0.70	209680
Lodgepole Pine (2)	0.82	0.56	0.67	281141
Ponderosa Pine (3)	0.67	0.65	0.66	33594
Cottonwood/Willow (4)	0.11	0.94	0.19	587
Aspen (5)	0.11	0.87	0.19	7333
Douglas-fir (6)	0.35	0.73	0.48	15207
Krummholz (7)	0.43	0.93	0.59	18350
accuracy			0.64	565892
macro avg	0.46	0.77	0.50	565892
weighted avg	0.74	0.64	0.67	565892



KNN

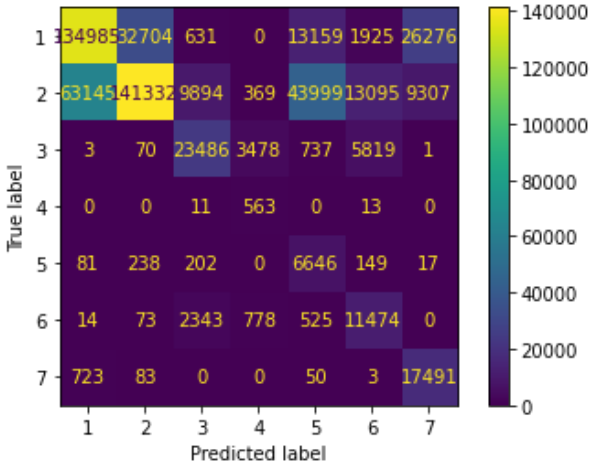
1. MyKNN

	precision	recall	f1-score	support
Spruce/Fir (1)	0.69	0.65	0.67	3798
Lodgepole Pine (2)	0.80	0.51	0.62	4866
Ponderosa Pine (3)	0.61	0.68	0.64	596
Cottonwood/Willow (4)	0.12	1.00	0.22	13
Aspen (5)	0.11	0.89	0.19	137
Douglas-fir (6)	0.36	0.73	0.49	278
Krummholz (7)	0.32	0.95	0.48	312
accuracy			0.60	10000
macro avg	0.43	0.77	0.47	10000
weighted avg	0.71	0.60	0.63	10000



2. KNeighborsClassifier

	precision	recall	f1-score	support
Spruce/Fir (1)	0.68	0.64	0.66	209680
Lodgepole Pine (2)	0.81	0.50	0.62	281141
Ponderosa Pine (3)	0.64	0.70	0.67	33594
Cottonwood/Willow (4)	0.11	0.96	0.19	587
Aspen (5)	0.10	0.91	0.18	7333
Douglas-fir (6)	0.35	0.75	0.48	15207
Krummholz (7)	0.33	0.95	0.49	18350
accuracy			0.59	565892
macro avg	0.43	0.77	0.47	565892
weighted avg	0.71	0.59	0.62	565892

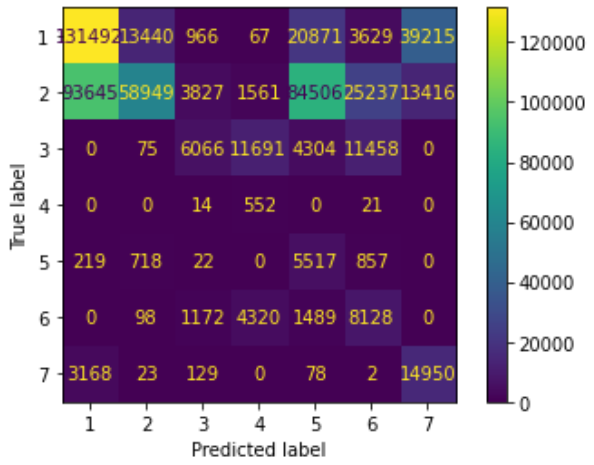


Naive Bayes

Важно на этом этапе вернуться к предобработке признаков и провести их трансформацию именно для этого метода

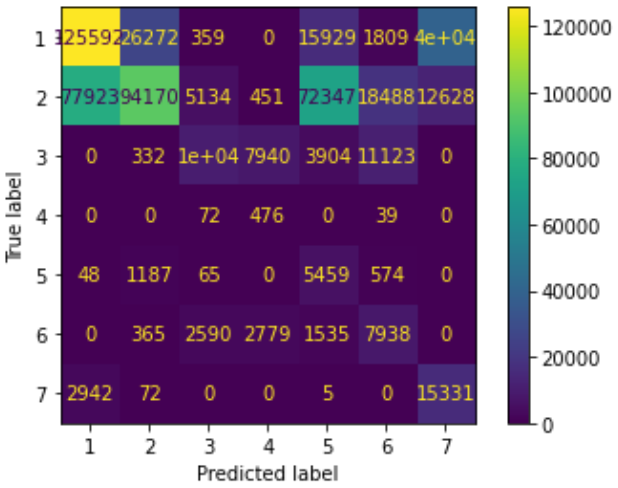
1. MyNaiveBayes

	precision	recall	f1-score	support
Spruce/Fir (1)	0.58	0.63	0.60	209680
Lodgepole Pine (2)	0.80	0.21	0.33	281141
Ponderosa Pine (3)	0.50	0.18	0.26	33594
Cottonwood/Willow (4)	0.03	0.94	0.06	587
Aspen (5)	0.05	0.75	0.09	7333
Douglas-fir (6)	0.16	0.53	0.25	15207
Krummholz (7)	0.22	0.81	0.35	18350
accuracy			0.40	565892
macro avg	0.33	0.58	0.28	565892
weighted avg	0.65	0.40	0.42	565892



2. GaussianNB

	precision	recall	f1-score	support
Spruce/Fir (1)	0.61	0.60	0.60	209680
Lodgepole Pine (2)	0.77	0.33	0.47	281141
Ponderosa Pine (3)	0.56	0.31	0.40	33594
Cottonwood/Willow (4)	0.04	0.81	0.08	587
Aspen (5)	0.06	0.74	0.10	7333
Douglas-fir (6)	0.20	0.52	0.29	15207
Krummholz (7)	0.23	0.84	0.36	18350
accuracy			0.46	565892
macro avg	0.35	0.59	0.33	565892
weighted avg	0.65	0.46	0.50	565892



3. Анализ результатов.

	Accuracy	Recall	Precision
MyMulticlassLogisticRegression	0.52	0.52	0.70
LogisticRegression	0.58	0.58	0.70
MySVM	0.51	0.51	0.65
SVC	0.64	0.64	0.74
MyKNN	0.60	0.60	0.71
KNeighborsClassifier	0.59	0.59	0.71
MyNaiveBayes	0.40	0.40	0.65
GaussianNB	0.46	0.46	0.65

Лучшей моделью по всем параметрам оказалась SVC, а из моих моделей лучшей оказалась KNN. Хотя стоит отметить, что все модели имеют не очень в ысокие значения метрик, это связано по большей части с особенностями самого набора данных.

Лучше всего модели угадывают:

- `Lodgepole Pine (2)`
- `Spruce/Fir (1)`

а хуже всего:

- `Cottonwood/Willow (4)`
- `Aspen (5)`

4. Выводы.

В ходе данной лабораторной работы я смог реализовать четыре классических алгоритма машинного обучения: логистическую регрессию, метод опорных векторов, метод k ближайших соседей и наивный байесовский классификатор.

Данные реализации способны решать задачу многоклассовой классификации, в чем и состояла главная трудность. В особенности для метода SVM, для его корректной работы была применена идея с евклидовой проекцией на симплекс. Все реализации сравнивались в соответствующими моделями из sckit-learn. При обучения стало понятно, что с «настоящими» данными многие методы работают довольно долго, особенно на этапе подбора параметров, порой приходилось сокращать тренировочную выборку, чтобы сократить время ожидания. Также стало ясно, что для байесовского классификатора трансформация признаков должна немного отличаться, чтобы алгоритм работал корректно.

После того, как все модели были обучены я в основном обращал внимание на две метрики: точность и доля правильных ответов, которые важны для задачи классификации типов лесного покрытия. Лучшая точность составила 74% для модели SVC, что довольно неплохо, так как задача оказалась не такой уж и простой.